

Carbon Programming

By K. J. Bricknell

Publisher: Sams Publishing
Pub Date: October 18, 2001
ISBN: 0-672-32267-6
Pages: 1608

Copyright
About the Author
Tell Us What You Think!
Introduction—The Carbon Application Programming Interface
- The First Task—Learn the C Language and CodeWarrior IDE
- The Carbon Programming Phase
- General Structure of Carbon Programming
- What You Will Need
- Demonstration Programs
- Terminology
- Early Days

Chapter 1. System Software, PowerPC Run-Time Environment, Managing Memory, and Resources
- System Software
- PowerPC Run-Time Environment
- Managing Memory—Mac OS 8/9
- Managing Memory—Mac OS X Considerations
- Resources
- Main Memory Manager Data Types and Functions
- Main Resource Manager Constants, Data Types, and Functions
- Demonstration Program SysMemRes Listing
- Demonstration Program SysMemRes Comments

Chapter 2. The Classic Event Manager—Low-Level and Operating System Events
The Two Event Managers
Overview of the Classic Event Model
Obtaining Information About Events
Flushing the Operating System Event Queue
Handling Events
Handling Events in Alerts and Dialogs
The 'SIZE' Resource
Main Event Manager Constants, Data Types and Functions
Demonstration Program LowEvents Listing
Demonstration Program LowEvents Comments

Chapter 3. Menus
Introduction—Types of Menus
Pull-Down Menus
Hierarchical Menus
Pop-Up Menus
Menu Objects, Menu IDs and Item Numbers, Command IDs, and Menu Lists
Creating Your Application's Menus
Providing Help Balloons (Mac OS 8/9)
Changing Menu Item Appearance
Adding Items to a Menu
Associating Data with Menu Items
Handling Menu Choices
Hiding and Showing the Menu Bar
Accessing Menus from Alerts and Dialogs
Main Menu Manager Constants, Data Types, and Functions
Demonstration Program Menus1 Listing
Demonstration Program Menus1 Comments
Demonstration Program Menus2 Listing
Demonstration Program Menus2 Comments

Chapter 4. Introduction to Windows
Scope of This Chapter
Window Basics
Types of Windows
Window Regions
Controls and Control Lists
The Window List
The Graphics Port and the Window Object

Events in Windows
Creating Your Application's Windows
Managing Multiple Windows and Associated Data
Handling Events
Manipulating Windows
Providing Help Balloons (Mac OS 8/9)
Main Window Manager Constants, Data Types, and Functions
Demonstration Program Windows1 Listing
Demonstration Program Windows1 Comments

Chapter 5. Carbon and Universal Procedure Pointers
Introduction
The 68LC040 Emulator and the Mixed Mode Manager
Universal Procedure Pointers and Carbon
Creation/Disposal Functions Relevant to Demonstration Programs

Chapter 6. The Appearance Manager
History
The Appearance Manager
Main Constants, Data Types, and Functions
Demonstration Program Appearance Listing
Demonstration Program Appearance Comments

Chapter 7. Introduction to Controls
Introduction
Standard Controls
The Control Definition Function
The Basic Controls, Primary Group Boxes (Text Title Variant), and User Panes
Activating, Deactivating, Hiding, Showing, Enabling, and Disabling Controls
Visual Feedback from the Basic Controls
Embedding Controls
Getting and Setting Control Data
The Control Object
Creating Controls
Updating, Moving, and Removing Controls
Handling Mouse Events in Controls
Determining and Changing Control Settings, and Getting a Control's Kind
Moving and Resizing Scroll Bars
Scrolling Operations With Scroll Bars
Small Versions of Controls

Main Control Manager Constants, Data Types and Functions Relevant to the Basic Controls, Primary Group Box and User Panes
Data Types
Functions
Demonstration Program Controls1 Listing
Demonstration Program Controls1 Comments
Demonstration Program Controls2 Listing
Demonstration Program Controls2 Comments

Chapter 8. Dialogs and Alerts
Introduction
Alert Types, Modalities, and Levels
Dialog Types and Modalities
Window Types for Alerts and Dialogs
Content of Alerts and Dialogs
Removing Dialogs
Creating and Removing Alerts
Creating Dialogs
Items in Dialogs
Displaying Alerts and Dialogs
Handling Events in Alerts and Dialogs
Closing and Disposing of Dialogs
Creating Displaying and Handling Window-Modal (Sheet) Alerts and Dialogs
Balloon Help for Dialogs—Mac OS 8/9
Help Tags for Dialogs—Mac OS X
Main Dialog Manager Constants, Data Types and Functions
Data Types
Functions
Relevant Window Manager Functions (Mac OS X Only)
Demonstration Program DialogAndAlerts Listing
Demonstration Program DialogsAndAlerts Comments
DialogsAndAlerts.c
Sheets.c

Chapter 9. The Finder and the Application
Introduction
An Application's Required Resources
Application Signature, File Creator, and File Types
Icons for the Finder
File Reference ('FREF') Resources
The Bundle ('BNDL') Resource

The Version ('vers') Resource
Finder Icon Help ('hfdr') Resource—Mac OS 8/9
Missing Application and Application-Missing Resources—Mac OS 8/9
The 'plst' Resource and Information Property Lists
Application Launching by the Finder
The Finder and the Catalog File
Preferences, Temporary Items, and Trash Folders—Using FindFolder
Stationery Pads
Using Aliases
Mac OS 9 Packages and Mac OS X Bundles
Relevant Constants, Data Types, and Functions
Demonstration Programs

Chapter 10. Apple Events
Introduction
Apple Event Attributes and Parameters
Data Structures Within Apple Events
Handling Apple Events
Required Apple Events—Contents and Required Action
Main Apple Event Manager and Appearance Manager Constants, Data Types, and Functions Relevant to Required Apple Events and Appearance Manager Apple Events
Demonstration Program AppleEvents
Demonstration Program AppleEvents Comments

Chapter 11. QuickDraw Preliminaries
QuickDraw and Imaging
RGB Colors and Pixels
Color and the Video Device
Graphics Port
Translation of RGB Colors to Pixel Values
Graphics Devices and GDevice Structures
Other Graphics Managers
Coping with Multiple Monitors
Relevant QuickDraw Constants, Data Types, and Functions
Demonstration Program PreQuickDraw Listing
Demonstration Program PreQuickDraw Comments

Chapter 12. Drawing with QuickDraw
Introduction
The Coordinate Plane, Points, Rectangles, and Regions
The Graphics Pen, Foreground and Background Colors, Pixel Patterns and Bit Patterns, and Transfer Modes
Drawing Lines and Framed Shapes

Drawing Painted and Filled Shapes
Erasing and Inverting Shapes
Drawing Pictures
Drawing Text
Copying Pixel Images Between Graphics Ports
Scrolling Pixels in the Port Rectangle
Manipulating Rectangles and Regions
Highlighting
Drawing Other Graphics Entities
Saving and Restoring the Graphics Port Drawing State
Main QuickDraw Constants, Data Types and Functions
Relevant Appearance Manager Data Types and Functions
Demonstration Program QuickDraw Listing
Demonstration Program QuickDraw Comments

Chapter 13. Offscreen Graphics Worlds, Pictures, Cursors, and Icons
Offscreen Graphics Worlds
Pictures
Cursors
Icons
Main Constants, Data Types and Functions—Offscreen Graphics Worlds
Main Constants, Data Types and Functions—Pictures
Main Constants, Data Types and Functions—Cursors
Appearance Manager Constants, Data Types and Functions—Cursors
Main Constants, Data Types and Functions—Icons
Demonstration Program GworldPicCursIcn Listing
Demonstration Program GWorldPicCursIcn Comments

Chapter 14. More on Controls
Introduction
Control Descriptions
Small Versions of Controls
Idle Processing
Defining Your Own Key Filter Function
Defining Your Own Edit Text Validation Function
Defining Your Own User Pane Functions
Main Constants, Data Types and Functions
Demonstration Program Controls3 Listing
Demonstration Program Controls3 Comments

Chapter 15. Carbon Printing

The Carbon Printing Manager
Printer Drivers
Page and Paper Rectangles
Page Setup Dialogs and Print Dialogs
Printing Sessions—The PMPrintSession Object
The PMPageFormat and PMPrintSettings Objects
Printing a Document
Text on the Screen and the Printed Page
Customizing the Page Setup and Print Dialogs
Displaying Page Setup and Print Dialogs As Window-Modal (Sheet) Dialogs
Saving and Retrieving a Page Format Object
Printing from the Finder—Mac OS 8/9
Main Carbon Printing Manager Constants, Data Types and Functions
Demonstration Program CarbonPrinting Listing
Demonstration Program CarbonPrinting Comments

Chapter 16. More on Windows
Introduction
Floating Windows
Utility and Toolbar Windows
Window Proxy Icons
Window Path Pop-Up Menus
Transitional Window Animation and Sounds
Creating and Storing Windows
Accessing Window Information
Moving and Positioning Windows
Associating Data with Windows
Adding To and Removing From the Update Region
Setting Content Region Color and Pattern
Window Scrolling
The Window Menu
Live Window Resizing
Main Constants, Data Types, and Functions
Demonstration Program Windows2 Listing
Demonstration Program Windows2 Comments

Chapter 17. The Carbon Event Manager
Overview
Event Reference, Class, and Type
Standard Event Handlers
The Application's Event Handlers

Sending and Explicitly Propagating Events
Event Timers
Getting Event Time
Other Aspects of the Carbon Event Model
Main Constants, Data Types, and Functions
Relevant Window Manager Constants and Functions
Demonstration Program CarbonEvents1 Listing
Demonstration Program CarbonEvents1 Comments
Demonstration Program CarbonEvents2 Listing
Demonstration Program CarbonEvents2 Comments

Chapter 18. Files and Navigation Services
Introduction
Files
Navigation Services
Main File Manager Constants, Data Types and Functions
Relevant Resource Manager Functions
Relevant Finder Interface Functions
Main Navigation Services Constants, Data Types, and Functions
Demonstration Program Files Listing
Demonstration Program Files Comments

Chapter 19. More on Resources and Core Foundation Preferences
Introduction
Search Path for Resources
Releasing and Detaching Resources
Creating, Opening, and Closing Resource Forks
Reading and Manipulating Resources
Writing Resources
Partial Resources
Application Preferences
Main Resource Manager Constants, Data Types and Functions
Main Core Foundation Preferences Services Functions
Demonstration Program MoreResources Listing
Demonstration Program MoreResources Comments

Chapter 20. Carbon Scrap
The Carbon Scrap Manager and the Scrap
Private Scrap
TextEdit, Dialogs, and Scrap
Main Carbon Scrap Manager Constants, Data Types, and Functions

Demonstration Program CarbonScrap Listing
Demonstration Program CarbonScrap Comments

Chapter 21. Text, Textedit, Dates, Times, and Numbers
Introduction
More on Text
Introduction to TextEdit
Monostyled TextEdit
Multistyled TextEdit
Formatting and Displaying Dates, Times, and Numbers
Main TextEdit Constants, Data Types and Functions
Main Constants, Data Types and Functions Relating to Dates, Times and Numbers
Demonstration Program MonoTextEdit Listing
Demonstration Program MonoTextEdit Comments
Demonstration Program DateTimeNumbers Listing
Demonstration Program DateTimeNumbers Comments

Chapter 22. Lists and Custom List Definition Functions
Introduction to Lists
Appearance and Features of Lists
Selection of Cells Using the Mouse
Selection of Cells Using the Keyboard
Creating, Disposing of, and Managing Lists
The List Box Control
Custom List Definition Functions
Main List Manager Constants, Data Types, and Functions
Relevant Text Utilities Data Type and Functions
Demonstration Program Lists Listing
Demonstration Program Lists Comments

Chapter 23. Drag and Drop
Overview
Drag Items, Drag Item Flavors, and Handlers
The Drag Region and The Drag Image
Drag Functions—Overriding Drag Manager Default Behavior
Main Drag Manager Constants, Data Types, and Functions
Relevant TextEdit Function
Demonstration Program Drag Listing
Demonstration Program Drag Comments

Chapter 24. Basic Sound and Speech

Introduction to Sound
Sound Production
Sound Recording—Mac OS 8/9
Speech
Relevant Constants, Data Types, and Functions
The AsynchSoundLib Library
Demonstration Program SoundAndSpeech Listing
Demonstration Program SoundAndSpeech Comments

Chapter 25. Miscellany
Notification from Applications in the Background
Progress Bars, Command-Period Key-Down Events, and Mouse-Down Events
The Color Picker
Coping with Multiple Monitors
Help Tags
Ensuring Compatibility with the Operating Environment
Carbon and Available APIs
Main Notification Manager Data Types and Functions
Relevant Process Manager Data Types and Functions
Relevant Event Manager Function
Relevant Color Picker Utilities Function
Relevant QuickDraw Function
Relevant Window Manager Function
Main Help Package Constants, Data Types, and Functions
Relevant Gestalt Manager Function
Demonstration Program Miscellany Listing
Demonstration Program Miscellany Comments

Chapter 26. More on Text—Multilingual Text Engine
Introduction
Working with MLTE
Main Constants, Data Types, and Functions
Demonstration Program MLTETextEditor Listing
Demonstration Program MLTETextEditor Comments

Copyright

Copyright © 2002 by Sams Publishing

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

Library of Congress Catalog Card Number: 2001095005

Printed in the United States of America

First Printing: November 2001

04 03 02 01 4 3 2 1

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Credits

Executive Editor

Jeff Schultz

Acquisitions Editor

Loretta Yates

Development Editor

Damon Jordan

Managing Editor

Charlotte Clapp

Indexer

Ron Strauss

Technical Editor

Russell Schelby

Team Coordinator

Amy Patton

Interior Designer

Gary Adair

Cover Designer

Bill Thomas

Page Layout

Mark Walchle

Dedication

To Janice, Samantha, Stephen, Bessie, and Kellie.

About the Author

K. J. Bricknell is a retired officer of the Royal Australian Air Force. His service career included flying duties in fighter squadrons and flying training units, together with staff duties associated with operational requirements, air training, and Air Force policy and plans at the Australian Department of Defense. His commands included No 3 (Fighter) Squadron and the Aircraft Research and Development Unit. An amateur programmer for many years, he is the author of *Macintosh C: A Hobbyist's Guide to Programming the Macintosh in C*, which was published on the Internet in 1996.

Tell Us What You Think!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

As an Executive Editor for Sams Publishing, I welcome your comments. You can e-mail or write me directly to let me know what you did or didn't like about this book-as well as what we can do to make our books stronger.

Please note that I cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail I receive, I might not be able to reply to every message.

When you write, please be sure to include this book's title and author as well as your name and phone or fax number. I will carefully review your comments and share them with the author and editors who worked on the book.

E-mail:	feedback@samspublishing.com
Mail:	Jeff Schultz Sams Publishing 201 West 103rd Street Indianapolis, IN 46290 USA

Introduction-The Carbon Application Programming Interface

Apple announced the intended introduction of a new operating system, to be known as Mac OS X, at the 1998 World Wide Developers Conference. Mac OS X, the first version of which was released on 24 March 2001, is not just another Mac OS update; it is a completely new operating system complete with "modern" operating system features such as pre-emptive multitasking and protected memory. It features a completely new user interface, called Aqua, whose appearance and behavior differs significantly from that of the original Mac OS (represented in its latest, and no doubt last, incarnation by Mac OS 9).

Mac OS X runs on G3 and G4 PowerPC machines only, meaning that machines based on PowerPC 604 and 603 microprocessors must necessarily remain with Mac OS 9 and earlier. A large installed base of these latter machines will no doubt remain for many years to come. In addition, it is likely that many owners of machines capable of running Mac OS X will nonetheless remain with Mac OS 9 and earlier. In these circumstances, it was perceived as all but essential that programs written to take advantage of Mac OS X's advanced features also be capable of running on Mac OS 8 and 9 without modification. Fortunately, Apple has devised the means whereby this can be achieved, namely, the Carbon Application Programming Interface (API).

An API is basically a collection of system software routines, two examples being `GetNewCWindow` (which programmers call to create windows) and `FSpOpenDF` (which programmers call to open a file). Mac OS 9 includes a collection of more than 8,000 such system software routines, which are now known collectively as the Classic API. Apple determined that about 2000 of these APIs were incompatible with a modern operating system like Mac OS X. The remaining 6,000 or so "clean" APIs were isolated and, together with a many additions, included in the Carbon API.

It turns out, therefore, that experienced Macintosh programmers will not need to learn a completely new API in order to program for the completely new operating system that is Mac OS X. Any programmer familiar with the Classic API will be able to transition to the Carbon API with comparatively little effort. In addition, newcomers to Macintosh programming will not be required to learn two APIs in order to write applications intended to run native on both Mac OS X and Mac OS 8 and 9.

That said, there is another API, called Cocoa, that may be used to write programs for Mac OS X. Cocoa's advantage is that it allows applications to be written more quickly than is the case using the Carbon API; however, such applications will not run on Mac OS 8 and 9.

A further advantage of Carbon is that existing applications written using the Classic API can be "carbonized" with but a fraction of the effort that would be required to completely rewrite them for Mac OS X using the Cocoa API.

This book, then, is for those who wish to learn to create applications that will run on both Mac OS X and Mac OS 8 and 9 and take advantage of Mac OS X's unique features when run on that system. It is intended as a reasonably comprehensive entry point to programming the Macintosh using the Carbon API.

The First Task-Learn the C Language and CodeWarrior IDE

The main assumption made by this book is that you have already learned both the C language and the Metrowerks CodeWarrior IDE (Integrated Development Environment) (http://www.metrowerks.com/platforms/mactools/).

CodeWarriorU (http://www.codewarrioru.com/CodeWarriorU) conducts on-line courses in the C language and the CodeWarrior IDE.

The Carbon Programming Phase

When you have learned the C language and the fundamentals of the CodeWarrior IDE, you are ready to open this book. As you move through this second phase of the journey, you will quickly discover that learning C and the IDE was by far the easiest part!

Essentially, this book covers all of the territory which, in the author's judgment, needs to be covered before you write your first serious application. This includes, for example, how to create and manage all elements of the user interface (menus, windows, controls, dialogs, alerts, lists, drag and drop), how to perform file input/output, how to print documents, how to draw text and graphics, and so on.

General Structure of Carbon Programming

The general structure of most chapters in this book is the same: first comes the information; then a list of constants, data types, and functions relevant to the subject of that chapter; then the source code listing of one or more demonstration programs related to the subject of that chapter; and, finally, line-by-line comments that explain the workings of the source code.

The book itself is supported by the Web site, which contains the CodeWarrior project and source code files, and Resorcerer resource files, for all demonstration programs. The demonstration programs associated with Chapters 18 to 25 exist in two versions:

- The primary version, which utilizes the Carbon event model introduced at Chapter 17, is the version whose source code listing and associated comments are included in the book itself.
- The secondary version utilizes the Classic event model (see Chapter 2). The explanatory comments for this version are contained in PDF files included in the relevant demonstration program (Classic event model version) folders.

What You Will Need

System Software

For Mac OS 8 and 9, Carbon Programming requires

- CarbonLib 1.4 or later. CarbonLib is the extension, located in the Extensions folder in the System folder, that implements the Carbon API on Mac OS 8 and 9.
- Mac OS 8.6 or later. CarbonLib 1.4 is not compatible with versions of Mac OS 8 earlier than 8.6.

For Mac OS X, Version 10.0.3 or later is required. (The so-called frameworks that implement the Carbon API on Mac OS X are an integral part of Mac OS X and are installed when you install that system. Frameworks are like shared libraries on Mac OS 8/9.)

Development System

As previously stated, it is assumed that your development system will be Metrowerks CodeWarrior. CodeWarrior Pro 6 (or later) is itself a Carbon application, and is much to be preferred.

Resource Editor

A resource editor allows you to create resources for programs and files. The resource editor you will need is Resorcerer, a product of Mathemaesthetics (http://www.mathemaesthetics.com/index.html).

References and Other Sources of Information

References enable you to access information relating to the system software, and are quite indispensable. The Carbon Developer Documentation section and the Mac OS 8 and 9 Developer Documentation Function Index (http://developer.apple.com/techpubs/macosx/Carbon/carbon.html and http://developer.apple.com/techpubs/FunctionIndex/FunctionIndex.html) at Apple's WWW site are the most up-to-date sources of information available. You might also wish to subscribe to Apple's Carbon development mailing list (http://www.lists.apple.com/mailman/listinfo/carbon-development).

Programmer's Calculator

You will find a programmer's calculator very useful for converting between decimal, hexadecimal, and binary values, the shareware program CalcWorks (http://sitelink.net/jbrochusoftware) being ideal for that purpose.

Icon Editor

Versions of Resorcerer earlier than Version 2.4 did not facilitate the creation of 32-bit icons or the large (128-by-128 pixel) thumbnail icons utilized by Mac OS X. If your version of Resorcerer is earlier than Version 2.4, the shareware program Iconographer (http://www.mscape.com/products/iconographer.html) is possibly the best option for the creation of these icons.

Universal Headers, CarbonLib, and Stub Libraries

The demonstration programs on the Web site assume the use of Version 3.4.1 or later of the Universal Headers and, as previously stated, CarbonLib 1.4 or later.

The Universal Headers were introduced at the same time as the Power Macintosh. Among other things, they enabled programmers to write (Classic API) source code capable of being compiled as either 680x0 code or PowerPC code-hence the term "Universal."

If the version of CodeWarrior you are using includes an earlier version of the Universal Headers and/or the version of CarbonLib in your Mac OS 8/9 Extensions folder is earlier than 1.4 (or does not exist), you should download the CarbonLib SDK (Software Development Kit) Version 1.4GM (or later) from http://developer.apple.com/carbon/ and

- Replace the existing folder CIncludes in Metrowerks CodeWarrior / Mac OS Support / Universal / Interfaces with the one in the SDK
- Copy the extension CarbonLib from the SDK to the Mac OS 8/9 Extensions folder, replacing the existing version if it exists
- Copy the two stub libraries CarbonLib and CarbonFrameWorkLib from the SDK to Metrowerks CodeWarrior / Mac OS Support / Universal / Libraries / StubLibraries, replacing the existing versions if they exist

Human Interface Guidelines-Mac OS 8/9

Useful additions to your library when you get further down the track would be the publications

- Macintosh Human Interface Guidelines and Mac OS 8 Human Interface Guidelines (http://developer.apple.com/techpubs/mac/HIGuidelines/HIGuidelines-2.html).

- Inside Mac OS X-Aqua Human Interface Guidelines (http://developer.apple.com/techpubs/macosx/macosx.html).

Demonstration Programs

For Mac OS 8/9, you should simply copy the folder Carbon Programming Demos from the Web site to a convenient location on your hard drive. For Mac OS X, you should copy the folder titled Carbon Programming Demos to the Applications folder and the folder titled Carbon Programming Documents to the relevant user's Documents folder.

Before running a demonstration program, you should read the top section of the source code comments in the relevant chapter (or, for the post-Chapter 17 Classic event model versions, in the previously mentioned PDF files). For most programs, this explains what to do, what to expect, and what to note. Note that the comments sections occasionally contain additional information incidental to the main purpose of the demonstration.

As far as is possible, each demonstration program avoids making calls to system software routines that are only explained in a later chapter. However, achieving that ideal has not been possible in the demonstration programs associated with the earlier chapters. For example, the demonstration program associated with Chapter 1 must, of necessity, make calls to system software routines relating to windows (the subject of Chapter 4) and drawing in a graphics port (the subject of Chapter 12). Where this occurs, you should simply accept, on faith, that the associated source code does as is stated in the demonstration program comments section. The important thing is to concentrate on that part of the source code pertaining to the subject of the chapter with which the program is associated.

Terminology

All but the later volumes of the official Mac OS reference work (Inside Macintosh) are Pascal-oriented, reflecting the fact that system software routines were originally designed to be called from a Pascal program. Because of this historically cosy relationship between Pascal and the Mac OS 8/9 system software, the C programmer will often be confronted by Pascal terminology. For example

- The Pascal terms "procedure" and "record" are sometimes used in C-oriented programming books and other publications in a context where the C programmer would ordinarily expect the terms "function" and "structure."
- In C, a routine which returns a value and a routine which does not return anything are both called **functions.** In Pascal, a routine which returns a value is also called a **function;** however, a routine which does not return anything is called a **procedure.**
- The names of many system software data structures end in `Rec` (for record) rather than, say, `Struc` (for structure).
- The names of certain fields in many system software data structures end in `Proc` (for procedure) rather than, say, `Func` (for function).

As a reflection of the fact that the later additions to Apple's Inside Macintosh series of publications are C-oriented rather than Pascal-oriented, this book uses C terminology exclusively. Hence "structure" is used rather than "record" and "function" is used rather than "procedure."

Early Days

This book was sent to press during the early days of both Mac OS X and the Mac OS 8/9 Carbon implementation. At that time, both systems contained bugs which affected some of the demonstration programs. Known issues in this regard, for which the author begs the reader's indulgence, are listed in the ReadMe file on the Web site.

In these circumstances, the reader is advised to periodically check the Carbon Programming updates site at www.samspublishing.com for updated text and code.

K. J. Bricknell, AM
Canberra, Australia
brick@asiaonline.net.au

Chapter 1. System Software, PowerPC Run-Time Environment, Managing Memory, and Resources

IN THIS CHAPTER

- System Software
- PowerPC Run-Time Environment
- Managing Memory-Mac OS 8/9
- Managing Memory-Mac OS X Considerations
- Main Memory Manager Data Types and Functions
- Main Resource Manager Constants, Data Types, and Functions
- Demonstration Program SysMemRes Listing
- Demonstration Program SysMemRes Comments

System Software

All Macintosh applications make many calls, for many purposes, to **system software functions.** Such purposes include, for example, the creation of standard user interface elements such as windows and menus, the drawing of text and graphics, and the coordination of the application's actions with other open applications.

> **Note**
>
> The main *other* open application that an application needs to work with is the **Finder.** The Finder is not really part of the system software, although it is sometimes difficult to tell where the Finder ends and the system software begins.

The entire collection of system software functions is divided into functional groups known as **managers.** In Carbon, each manager is part of a higher-level grouping called a **manager group.** The manager groups and their constituent managers are shown in the following, in which those managers of particular relevance to this book appear in bold typeface:

Application Utilities Facilities for extending human interface features and data-sharing capabilities beyond those provided by the Human Interface Toolbox.

AltiVec Utilities	Find By Content	Translation Manager

Carbon Core Essential services that are generally devoid of a user interface, including memory and process management.

Alias Manager	**Collection Manager**	Component Manager	**File Manager**
Folder Manager	**Low Memory Accessors**	Memory Management Utilities	**Memory Manager**
Mixed Mode Manager	Multiprocessing Services	**Notification Manager**	Pascal String Utilities

Resource Manager	Thread Manager	Time Manager	

Common OS Services

Display Manager	**Gestalt Manager**	Power Manager	**Process Manager**
Speech Recognition Manager	**Speech Synthesis Manager**		

Core Foundation Abstracts several operating system utilities fundamental to both Mac OS 8/9 and Mac OS X.

Base Services	Bundle Services	Collection Services	Plug-in Services
Preferences Services	**Property List Services**	**String Services**	URL Services
Utility Services	XML Services		

Graphics Services Allows applications to construct and render images.

Carbon Printing Manager	**Color Picker Manager**	ColorSync Manager	HTML Rendering Library
Palette Manager	**QuickDraw Manager**		

Hardware Interfaces

Device Manager	PCI Card Services

Human Interface Toolbox Comprises groups of libraries that implement the Mac OS human interface.

Appearance Manager	**Apple Help**	**Carbon Event Manager**	**Carbon Help Manager**
Control Manager	**Dialog Manager**	**Drag Manager**	**Event Manager**
Finder Interface	**Icon Utilities**	**List Manager**	**Menu Manager**
Navigation Services	**Picture Utilities**	**Scrap Manager**	**Window Manager**

Networking and Communication Services

Internet Config	NSL manager	Open Transport	URL Access Manager

QuickTime Assist your application in combining multiple forms of communication, such as text, pictures, video, and music.

Image Components Manager	Media Handlers	Movie Toolbox	QuickTime Compression
QuickTime Media Layer	QuickTime Music	QuickTime Streaming	QuickTime VR

Sound Manager

Runtime Services Provides programming interfaces that prepare code for execution and control how functions call each other.

Code Fragment Manager	Debugger Services	Math and Logical Utilities

Scripting and Apple Events

Apple Event Manager	Open Scripting Architecture

Text and Other International Services Assistance in creating applications for world-wide markets

Apple Type Services for Unicode Imaging	**Date, Time, and Measurement Utilities**	**Font Manager**
FontSync	**Multilingual Text Engine**	**QuickDraw Text**
Script Manager	Text Encoding Conversion Manager	Text Services Manager
Text Utilities	**TextEdit**	Unicode Utilities

PowerPC Run-Time Environment

A run-time environment is a set of conventions that determines how code is to be loaded into memory, where it is to be stored, how it is to be addressed, and how functions call other functions and system software routines. The system software and your development system jointly determine the run-time environment.

Fragments

The PowerPC run-time model is based on the use of **fragments** as the standard way of organizing executable code and data in memory. A fragment is any block of executable PowerPC code and its associated data. Fragments can be of any size, and are complete, executable entities.

There are three broad categories of fragments, namely, **applications, import libraries,** and **extensions.** (Import libraries and system extensions are sometimes called **shared libraries** or **dynamically-linked libraries.**) An application is a fragment that can be launched by the user from the Finder.

Code Fragment Resource

You use a code fragment (`'cfrg'`) resource (see Resources, later in this chapter) to specify some characteristics of a code fragment. For an application, the code fragment resource indicates to the Process Manager that the application's data fork contains an executable code fragment container.

You do not have to create the `'cfrg'` resource yourself because CodeWarrior does this for you when you compile your application.

Organization of Memory

The basic organization of memory in the PowerPC run-time environment is as follows:

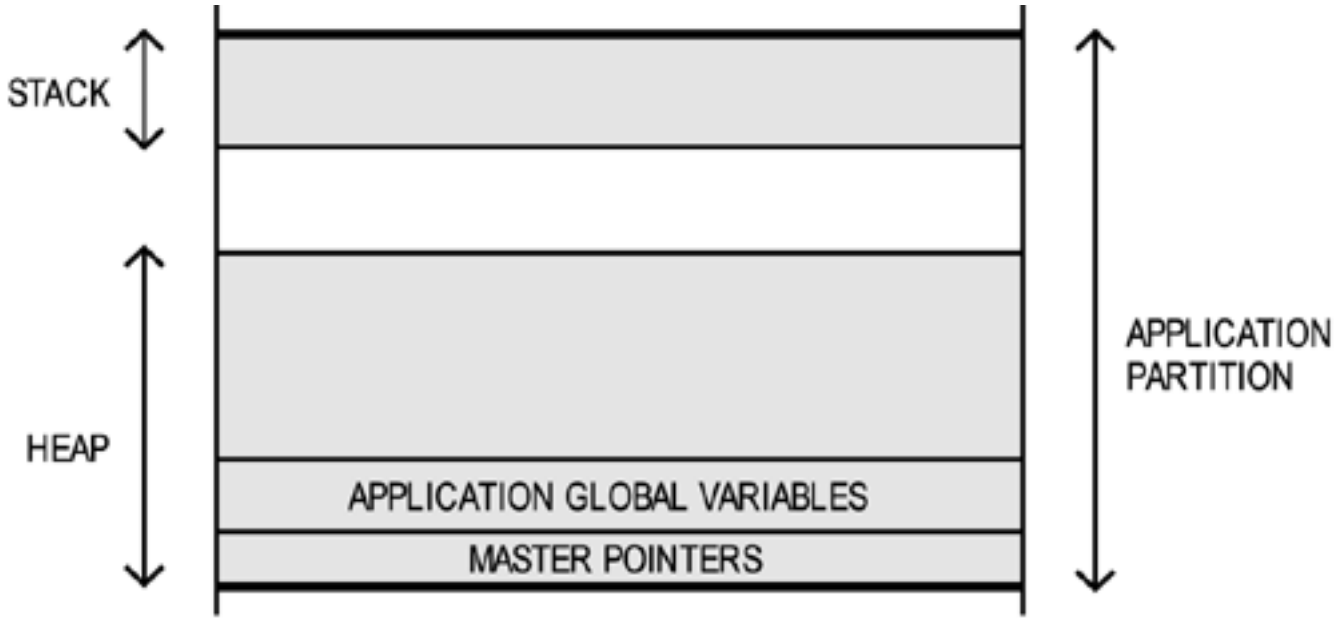
- A **system partition,** which is reserved for the exclusive use of the system, occupies the lowest memory address. Part of the system partition is occupied by the system global variables, most of which contain information of use to the system software, for example:
    - `Ticks`, which contains the number of ticks since system startup. (A tick is 1/60th of a second.)
    - `MBarHeight`, which contains the height of the menu bar.

- Most of the remaining space is allocated to the Process Manager, which creates an **application partition** for each open application. Application partitions are loaded into the top part of RAM first.

The Application Partition

In each application partition, there is a **stack** and a **heap,** as well as space for the application's **global variables** (see Fig 1).

Figure 1. STRUCTURE OF APPLICATION PARTITION



The Stack

The stack is used for memory allocation associated with the execution of functions. When an application calls a function, space is allocated on the stack for a **stack frame,** which contains the function's parameters, local variables and return address. The local variables and function parameters are popped off the stack when the function has executed. The C compiler generates the code that creates and deletes stack frames for each function call.

It is important to understand that the Memory Manager has no way of preventing the stack from encroaching on, and corrupting, the heap. Ordinarily, unless your application uses heavy recursion (one function repeatedly calling itself), you almost certainly will never need to worry about the possibility of stack overflow.

> **Note**
>
> The reason that recursion increases the risk is that, each time the function calls itself, a *new* copy of that function's parameters and variables is pushed onto the stack.

The Heap

The application heap is that part of the application partition in which memory is dynamically allocated and released on demand. Space within the heap is allocated, in blocks, by both direct or indirect calls to the Memory Manager. An indirect allocation arises from a call to a function which itself calls a Memory Manager memory allocation function.

Managing Memory-Mac OS 8/9

Inside the Application Heap-Nonrelocatable and Relocatable Memory Blocks

An application may use the Memory Manager to allocate two different types of memory blocks: a **nonrelocatable block** and a **relocatable block.**

Nonrelocatable Blocks

Nonrelocatable blocks are blocks whose location in the heap is fixed. In its attempts to avoid heap fragmentation, the Memory Manager allocates nonrelocatable blocks as low in the heap as possible, where necessary moving relocatable blocks upward to make space. Nonrelocatable blocks are referenced using a **pointer** variable of data type `Ptr`. `Ptr` is defined as follows:

```
typedef char *Ptr;  // A pointer to a signed char.
```

A pointer is simply the address of a byte in memory. Thus a pointer to a nonrelocatable block is simply the address of the first byte in that block of memory. Note that if a copy is made of the pointer variable after the block is created, and since the block cannot be moved, that copy will correctly reference the block until it is disposed of.

The Memory Manager function `NewPtr` allocates a nonrelocatable block, for example:

```
Ptr  myPointer;
myPointer = NewPtr(sizeof(myDataStructure));
```

Nonrelocatable blocks are disposed of by a call to `DisposePtr`.

Unlike relocatable blocks, there are only five things that your application can do with a nonrelocatable block: create it; obtain its size; resize it; find which heap zone owns it; dispose of it.

Relocatable Blocks

Relocatable blocks are blocks that can be moved within the heap-for example, during heap compaction operations. To reference relocatable blocks, the Memory Manager uses **double indirection,** that is, the Memory Manager keeps track of a relocatable block with a **master pointer,** which is itself part of a nonrelocatable **master pointer block** in the application heap. When the Memory Manager moves a relocatable block, it updates the master pointer so as to ensure that the master pointer always contains the current address of the relocatable block.

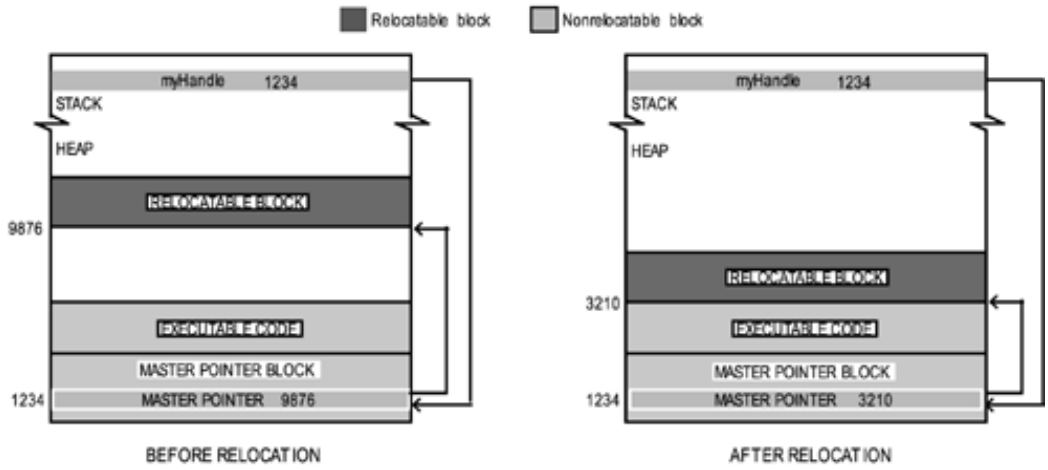
One master pointer block, which contains 64 master pointers, is allocated for your application by the Memory Manager at application launch. This block is located at the very bottom of the application heap (see Fig 1). The function `MoreMasterPointers` may be called by your application to allocate additional master pointers. To ensure that these additional (nonrelocatable) blocks are allocated as low in the heap as possible, the call to `MoreMasterPointers` should be made at the beginning of your program.

> **Note**
>
> If these calls are not made, the Memory Manager will nonetheless automatically allocate additional blocks during application execution if required. However, since master pointer blocks are nonrelocatable, such allocation, which will not be at the bottom of the heap, is a possible cause of heap fragmentation. Your call to MoreMasterPointers should thus allocate sufficient master pointers to ensure that the Memory Manager never needs to create additional master pointer blocks for you. (You can empirically determine how many master pointers to allocate using a low-level debugger.)

Relocatable blocks are referenced using a **handle** variable of data type `Handle`. A handle contains the address of a master pointer, as illustrated in Fig 2. `Handle` is defined as follows:

```
typedef Ptr *Handle;  // A pointer to a master pointer.
```

Figure 2. A HANDLE TO A RELOCATABLE BLOCK



The Memory Manager function `NewHandle` allocates a relocatable block, for example:

```
Handle myHandle;
myHandle = NewHandle(sizeof(myDataStructure));
```

A relocatable block can be disposed of by a call to `DisposeHandle`. Note, however, that the Memory Manager does not change the value of any handle variables that previously referenced that deallocated block. Instead, those variables still hold the address of what was once the relocatable block's master pointer. If you inadvertently use a handle to a block you have already disposed of, your application could crash. You can avoid these problems by assigning the value `NULL` to the handle variable after you dispose of the relocatable block.

Heap Fragmentation, Compaction, and Purging

The continuous allocation and release of memory blocks which occurs during an application's execution can result in a condition called **heap fragmentation.** The heap is said to be fragmented when nonrelocatable blocks or **locked relocatable blocks** (see the next section) are scattered about the heap, leaving "holes" of memory between those blocks.

The Memory Manager continuously attempts to create more contiguous free memory space through an operation known as **heap compaction,** which involves moving all relocatable blocks as low in the heap as possible. However, because the Memory Manager cannot move relocatable blocks "around" nonrelocatable blocks and locked relocatable blocks, such blocks act like log-jams if there is free space below them. In this situation, the Memory Manager may not be able to satisfy a new memory allocation request because, although there may be enough total free memory space, that space is broken up into small noncontiguous blocks.

Heap fragmentation would not occur if all memory blocks allocated by the application were free to move during heap compaction. However, there are two types of memory block which are not free to move: nonrelocatable blocks and relocatable blocks that have been temporarily locked in place.

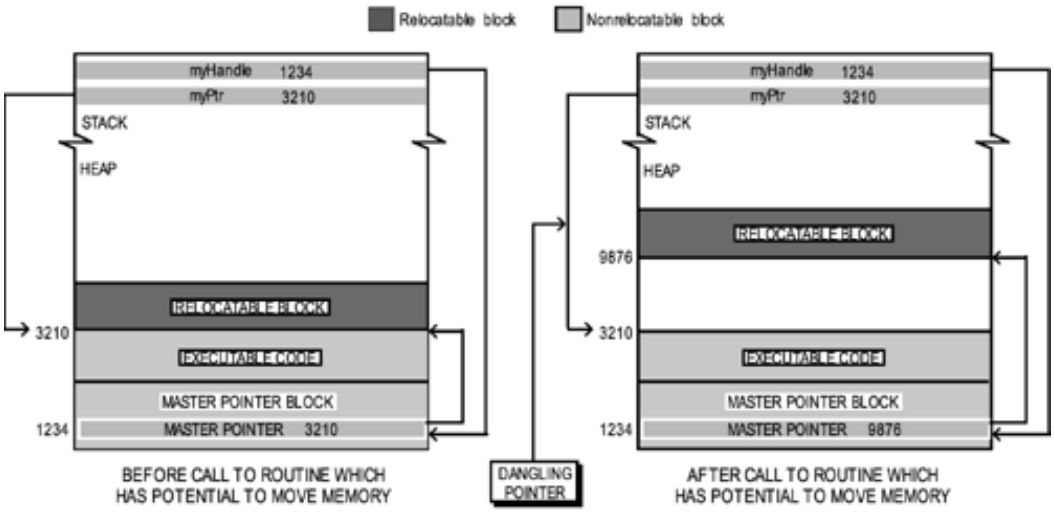
Locking and Unlocking Relocatable Blocks

Despite the potential of such action to inhibit the Memory Manager's heap compaction activities, it is nonetheless necessary to lock relocatable blocks in place in certain circumstances.

For example, suppose you dereference a handle to obtain a pointer (that is, a *copy* of the master pointer) to a relocatable block and, for the sake of increased speed, use that pointer within a loop to read or write data to or from the block. If, within that loop, you call a function that has the potential to move memory, and if that function actually causes the relocatable block to be

moved, the master pointer will be correctly updated but your copy (the pointer) will not. The net result is that your pointer no longer points to the data and becomes what is known as a **dangling pointer.** This situation is illustrated at Fig 3.

Figure 3. A DANGLING POINTER



Note

Accessing a relocatable block by double indirection (that is, through its handle) instead of by single indirection (ie, through its master pointer) requires an extra memory reference.

The documentation for system software functions indicates whether a particular function has the potential to move memory. Generally, any function that allocates space from the application heap has this potential. If such a function is not called in a section of code, you can safely assume that all blocks will remain stationary while that code executes.

Relocatable blocks may be locked and unlocked using `HLock` and `HUnlock`. The following example illustrates the use of these functions.

```
typedef struct
{
  short intArray[1000];
  char  ch;
}  Structure, *StructurePointer, **StructureHandle;

void  myFunction(void)
{
  StructureHandle   theHdl;
  StructurePointer  thePtr;
  short             count;

  theHdl = (StructureHandle) NewHandle(sizeof(Structure));

  HLock(theHdl);                    // Lock the relocatable block ...
  thePtr = *theHdl;                 // because the handle has been dereferenced ...

  for(count=0;count<1000;count++)
  {
    (*thePtr).intArray[count] = 0;  // and used in this loop ...
    DrawChar((char)'A');            // which calls a function which could cause
  }                                  // the relocatable block to be moved.

  HUnlock(theHdl);                  // On loop exit, unlock the relocatable block.
```

```
}
```

Moving Relocatable Blocks High

The potential for a locked relocatable block to contribute to heap fragmentation may be avoided by moving the block to the top of the heap before locking it. This should be done if new nonrelocatable blocks are to be allocated while the relocatable block in question is locked.

`MoveHHi` is used to move relocatable blocks to the top of the heap. `HLockHi` is used to move relocatable blocks to the top of the heap and then lock them. Be aware, however, that `MoveHHi` and `HLockHi` cannot move a block to the top of the heap if a nonrelocatable block or locked relocatable block is located between its current location and the top of the heap. In this situation, the block will be moved to a location immediately below the nonrelocatable block or locked relocatable block.

Purging and Reallocating Relocatable Blocks

In addition to compacting the heap in order to satisfy a memory allocation request, the Memory Manager may **purge** unlocked relocatable memory blocks that have been made purgeable.

`HPurge` and `HNoPurge` change a relocatable block from unpurgeable to purgeable and vice versa. When you make a relocatable block purgeable, your program should subsequently check the handle to that block before using it if calls are made to functions which could move or purge memory.

If the handle's master pointer is set to `NULL`, then the Operating System has purged its block. To use the information formerly in the block, space must then be reallocated for it and its contents must be reconstructed.

Effect of a Relocatable Block's Attributes

Two **attributes** of a relocatable block are whether the block is currently locked/unlocked or purgeable/non-purgeable. These attributes are stored in bits in the block's master pointer **tag byte.** The following summarizes the effect of these attributes on the Memory Manager's ability to move and/or purge a relocatable block:

Tag Byte Indicates Block Is		*The Memory Manager Can*	
Locked	***Purgeable***	***Move The Block***	***Purge the Block***
NO	**NO**	**YES**	**NO**
NO	**YES**	**YES**	**YES**
YES	NO	NO	NO
YES	**YES**	NO	NO

> **Note**
>
> The tag byte is the high byte of a master pointer. If Bit 5 of the tag byte is set, the block is a resource block. If Bit 6 is set, the block is purgeable. If Bit 7 is set, the block is locked.

Note that a relocatable block created by a call to `NewHandle` is created initially unlocked and unpurgeable, and that locking a relocatable block will also make it unpurgeable if it is currently purgeable.

Avoiding Heap Fragmentation

The ideal heap is one with all nonrelocatable blocks at the bottom of the heap, all unlocked relocatable blocks above that, free space above that, and all relocatable blocks which must be locked for significant periods at the top of the heap. This ideal can be approached, and significant heap fragmentation avoided, by adherence to the following rules:

- At the beginning of the program, call `MoreMasterPointers` to allocate at least as many (nonrelocatable) master pointers as are required during program execution.
- Allocate all other required nonrelocatable blocks at the beginning of the application's execution.
- Avoid disposing of, and subsequently reallocating, nonrelocatable blocks during the application's execution.
- If you need to allocate a relocatable block that will need to be locked for a long period of time, call `ReserveMem` at the beginning of the program to reserve memory for the block as close to the bottom of the heap as is possible, and then lock the block immediately after allocating it.
- If a relocatable block is to be locked for a short period of time and nonrelocatable blocks are to be allocated while it is locked, call `MoveHHi` to move the relocatable block to the top of the heap and then lock it. Unlock the block when it no longer needs to be locked.

Also bear in mind that, in memory management terms, a relocatable block that is always locked is worse than a nonrelocatable block in that nonrelocatable blocks are always allocated as low in the heap as possible, whereas a relocatable block is allocated wherever the Memory Manager finds it convenient.

Master Pointer Tag Byte-HGetState and HSetState

There are certain circumstances where you will want to save, and later restore, the current value of a relocatable block's master pointer tag byte. Consider the following example, which involves three of an imaginary application's functions, namely, Function A, Function B, and Function C:

- Function A creates a relocatable block. For reasons of its own, Function A locks the block before executing a few lines of code. Function A then calls Function C, passing the handle to that function as a formal parameter.
- Function B also calls Function C at some point, passing the relocatable block's handle to it as a formal parameter. The difference in this instance is that, due to certain machinations in other areas of the application, the block is unlocked when the call to Function C is made.
- Function C, for reasons of its own, needs to ensure that the block is locked before executing a few lines of code, so it makes a call to `HLock`. Those lines executed, Function C then unlocks the block before returning to the calling function. This will not be of great concern if the return is to Function B, which expects the block to be still unlocked. However, if the return is to Function A, and if Function A now executes some lines of code which assume that the block is still locked, disaster could strike.

This is where the Memory Manager functions `HGetState` and `HSetState` come in. The sequence of events in Function C should have been as follows:

```
SInt8 theTagByte;
...
theTagByte = HGetState(myHandle);  // Whatever the current state is, save it.
HLock(myHandle);                   // Redundant if Function A is calling, but no harm.

(Bulk of the Function C code, which requires handle to be locked.)

HSetState(myHandle,theTagByte)     // Leave it the way it was found. (It could have
                                   // been locked. It could have been unlocked.)
return;
```

This is an example of what might be called a "well-mannered function". It is an example of a rule that you may wish to apply whenever you write a function that takes a handle to a relocatable block as a formal parameter: If that function calls `HLock`, make sure that it leaves the block's tag byte (and thus the locked/unlocked bit) in the condition in which it found it.

> **Note**
>
> Of course, this save/restore precaution will not really be necessary if you are absolutely certain that the block in question will be in a particular state (locked or unlocked) every time Function C is called. But there is nothing wrong with a little coding overkill to protect yourself from, for example, some future source code modifications which may add other functions which call Function C, and which may assume that the block's attributes will be handed back in the condition in which Function C found them.

Memory Leaks

When you have no further use for a block of memory, you ordinarily return that memory to the Memory Manager by calling `DisposePtr` or `DisposeHandle` (or `ReleaseResource`). In certain circumstances, not disposing of a block that is no longer required can result in what is known as a **memory leak.**

Memory leaks can have unfortunate consequences for your application. For example, consider the following function:

```
void  theFunction(void)
{
  Ptr   thePointer;
  OSErr osError;

  thePointer = NewPtr(10000);
  if(MemError() == memFullErr)
    doErrorAlert(eOutOfMemory);

  // The nonrelocatable block is used for some temporary purpose here, but is not
  // disposed of before the function returns.
}
```

When `theFunction` returns, the 10,000-byte nonrelocatable block will still exist (even though, incidentally, the local variable which previously pointed to it will not). Thus a large nonrelocatable block for which you have no further use remains in memory (at what is now, incidentally, an unknown location). If `theFunction` is called several more times, a new nonrelocatable block will be created by each call and the size of the memory leak will grow, perhaps eventually causing `MemErr` to return `memFullErr`. In this way, memory leaks can bring your application to a standstill and might, in some circumstances, cause it to crash.

> **Note**
>
> The dynamic memory inspection tool ZoneRanger, which is included with Metrowerks CodeWarrior, can be used to check your application for memory leaks.

Memory Manager Errors

The error code resulting from the last call to a Memory Manager function might be retrieved by calling the function `MemError`. Some of the error codes which may be returned by `MemError` are as follows:

Value	*Constant*	*Description*
`0`	`noErr`	No error occurred.
`-108`	`memFullErr`	No room in heap.
`-109`	`nilHandleErr`	Illegal operation on a `NULL` handle.
`-117`	`memLockedErr`	Trying to move a locked block (`MoveHHi`).

Managing Memory-Mac OS X Considerations

Preamble-Memory in Mac OS X

In Mac OS X, each application runs in its own address space, meaning that an application cannot reference memory locations outside its assigned address space.

Compared with Mac OS 8/9, Mac OS X uses an entirely different heap structure and allocation behavior.

Mac OS X uses a dynamic and highly efficient virtual memory system, which is always enabled. Carbon application must therefore assume that virtual memory is always on.

Memory Management Considerations

Allocating Memory

The functions `FreeMem` (get the amount of free memory in the current heap zone), `PurgeMem` (purge blocks without compacting the heap), and `MaxMem` (compact the heap and purge all the purgeable blocks from the current heap zone) are all included in Carbon, and behave as expected under Mac OS 8/9. When your application is running on Mac OS X, however, these functions are more or less meaningless because, in Mac OS X, the system provides virtually unlimited memory.

Stack Size

A Carbon application may have different stack sizes under Mac OS 8/9 and Mac OS X; accordingly, the `StackSpace` function (which returns the amount of unused space on the stack at the time of the call) is no longer very useful.

Master Pointer Allocation

Master pointers do not need to be pre-allocated, or their number optimized, in the Mac OS X memory model. Accordingly, the effect of the `MoreMasterPointers` function only applies when your application is run on Mac OS 8/9.

Resources

In order to meet various requirements of the system software, your application must provide its own **resources,** for example, resources which describe the application's user interface elements such as menus, windows, controls, dialogs, and icons. In addition, the system software itself provides a number of resources (for example, fonts, patterns, icons, and so on) that may be used by your application.

The concept of resources reflects the fact that, in the Macintosh environment, intermixing code and data in a program is not encouraged. For example, it is usual practice to separate changeable data, such as message strings which may need to be changed in a foreign-language version of the application, from the application's code. All that is required in such a case is to create a resource containing the foreign language version of the message strings. There is thus no necessity to change and recompile the source code in order to produce a foreign-language version of the application.

The subject of resources is closely related to the subject of files. A brief digression into the world of files is thus necessary.

About Files-The Data Fork and the Resource Fork

On the Macintosh, a **file** is a named, ordered sequence of bytes stored on a volume and divided into two forks:

- ***The Data Fork.*** The data fork typically contains data created by the user.
- ***The Resource Fork.*** The resource fork of a file contains resources, which are collections of data of a defined structure and type.

Note

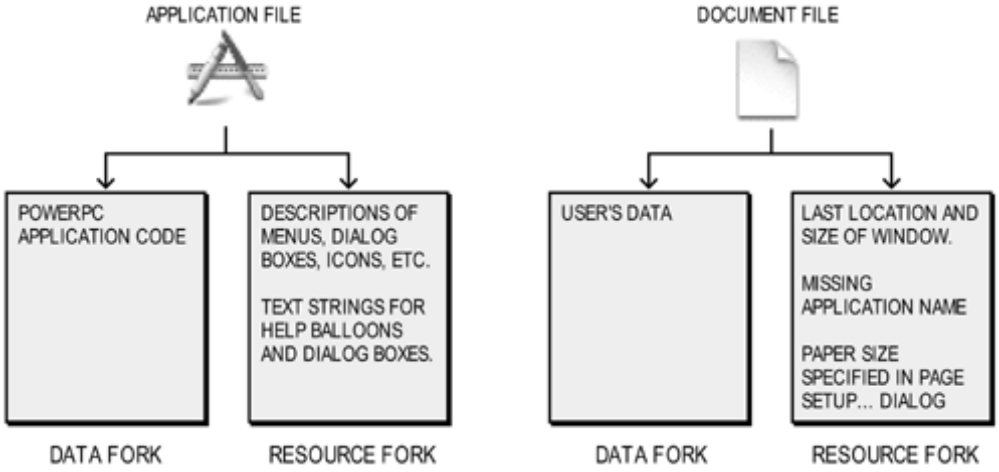
Before the introduction of Mac OS X and Carbon, application and document resources were invariably stored in the resource fork of, respectively, application and document files. Mac OS X and Carbon introduced an alternative scheme whereby resources can be placed in the data fork of a separate resource file. The main reason for this alternative scheme is that it prevents application and document files from losing resource data when moved around different file systems or between Macintosh and non-Macintosh systems.

This alternative scheme is, however, not addressed in this book. The content of this book assumes that application and document resources are stored in the resource fork of application and document files, and the associated demonstration program files reflect that arrangement.

All Macintosh files contain both a resource fork and a data fork, even though one or both of these forks may be empty. Note that the resource fork of a file is also called a **resource file,** because in some respects you can treat it as if it were a separate file.

The resource fork of a document file contains resources specific to the document, such as the size and location of the document's window when the document was last closed. The resource fork of an application file includes, typically, resources that describe the application's windows, menus, and so on. Fig 4 illustrates the typical contents of the data and resource forks of an application file and a document file.

Figure 4. TYPICAL CONTENTS OF DATA FORKS AND RESOURCE FORKS IN APPLICATION AND DOCUMENT FILES



The data fork can contain any kind of data organized in any fashion. Your application can store data in the data fork of a document file in whatever way it deems appropriate, but it needs to keep track of the exact byte location of each particular piece of saved data in order to be able to access that data when required. The resource fork, on the other hand, is highly structured. As will be seen, all resource forks contain a map which, amongst other things, lists the location of all resources in the resource fork. This greatly simplifies the task of accessing those resources.

Resources and the Application

During its execution, an application might read resources from

- The application's resource file, which is opened automatically when the application is launched.
- The System file, which is opened by the Operating System at startup and which contains resources that are shared by all applications (for example, fonts, icons, sounds, and so on) and resources that applications might use to help present the standard user interface.
- Other resource files, such as a preferences file in the Preferences folder holding the user's application-specific preferences, or the resource fork of a document file, which might define certain document-specific preferences.

The Resource Manager provides functions which allow your application to read in these resources and, in addition, to create, delete, open, modify and write resources in, from and to any Macintosh file. The following, however, is concerned only with creating resources for the application's resource file and with reading in **standard resources** from the application and System files. Other aspects of resources, including **custom resources** and resources in files other than the application and System files, are addressed in Chapter 19.

Resource Types and Resource IDs

An application refers to a resource by passing the Resource Manager a **resource specification,** which consists of the **resource type** and a **resource ID:**

- ***Resource Type***. A resource type is specified by any sequence of four alphanumeric characters, including the space character, which uniquely identifies a specific type of resource. Both uppercase and lowercase characters are used. Some of the standard resource types defined by the system software are as follows:

Type	*Description*	*Type*	*Description*
`'ALRT'`	Alert template.	`'DLOG'`	Dialog template.
`'DITL'`	Item list in dialog or alert.	`'SIZE'`	Size of application's partition and other info.
`'FONT'`	Bitmapped font.	`'WIND'`	Window template.
`'MBAR'`	Menu bar.	`'snd '`	Sound. (Space character is required.)
`'PICT'`	QuickDraw picture.	`'STR#'`	String list.

  You can also create your own custom resource types if your application needs resources other than the standard types. An example would be a custom resource type for application-specific preferences stored in a preferences file.

Note

When choosing the characters to identify your custom resource types, note that Apple reserves for its own use resource types consisting entirely of lowercase characters and special symbols. Your custom resource types should therefore contain at least one uppercase character.

- ***Resource ID.*** A resource ID identifies a specific resource of a given type by number. System resource IDs range from -32768 to 127. In general, resource IDs from 128 to 32767 are available for resources that you create yourself, although the numbers you can use for some types of resources (for example, font families) are more restricted. An application's definition functions should use IDs between 128 and 4095.

Creating a Resource

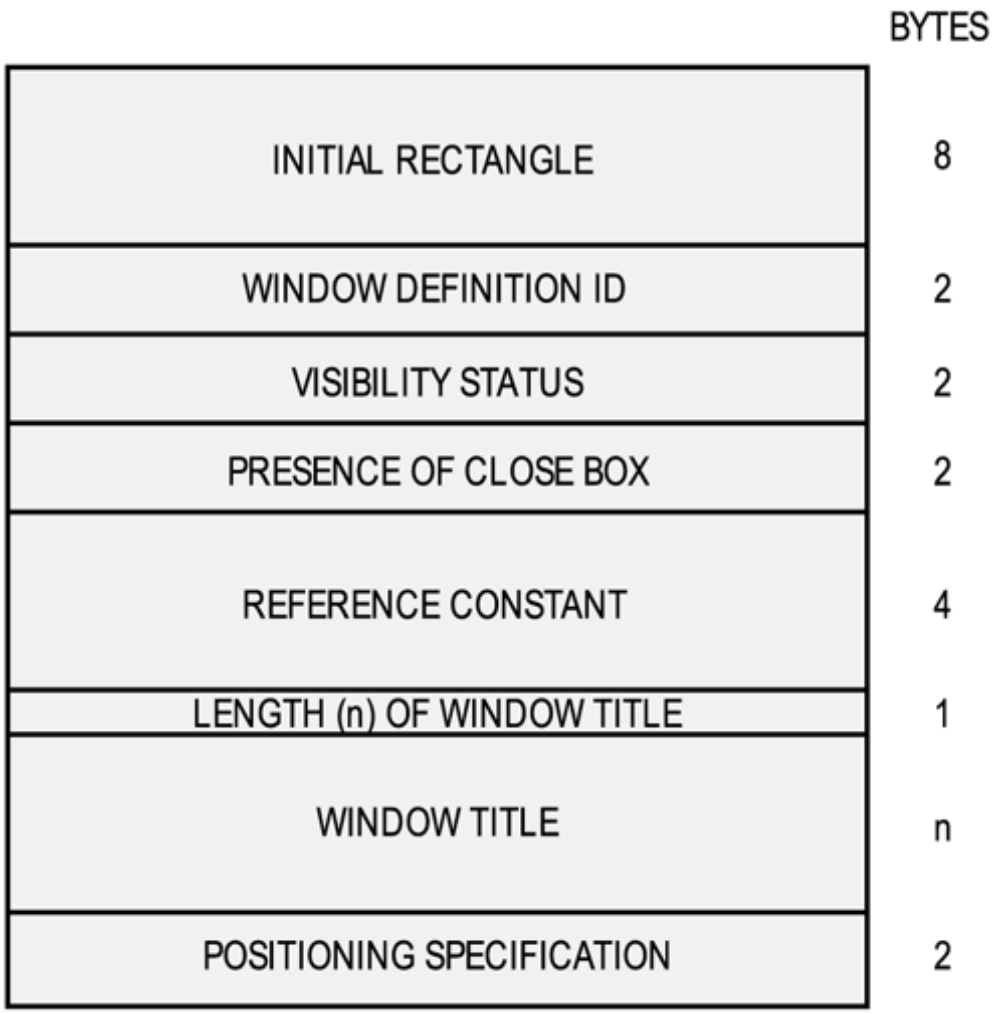
At the very least, you may need to create resources for the standard user interface elements used by your application. You typically define the user interface elements in resources and then use Menu Manager, Window Manager, Dialog Manager or Control Manager functions to create these elements, based on their resource descriptions, as needed.

You can create resource descriptions using a resource editor such as Resorcerer (which uses the familiar point-and-click approach), or you can provide a textual, formal description of resources in a file and then use a resource compiler, such as Rez, to compile the description into a resource. An example of a resource definition for a window in Rez input format is as follows:

```
resource 'WIND' (128, preload, purgeable)
{
  { 64,60,314,460} ,           /* Window rectangle. (Initial window size and location.) */
  kWindowDocumentProc,       /* Window definition ID. */
  invisible,                 /* Window is initially invisible. */
  goAway,                    /* Window has a close box. */
  0x0,                       /* Reference constant. */
  "untitled",                /* Window title. */
  staggerParentWindowScreen  /* Optional positioning specification. */
} ;
```

The structure of the compiled `'WIND'` resource is shown at Fig 5.

Figure 5. STRUCTURE OF A COMPILED WINDOW ('WIND') RESOURCE



Note

This book assumes the use of Resorcerer, and all demonstration program resources were created using Resorcerer.

Resource Attributes

Note the words `preload` and `purgeable` in the preceding `'WIND'` resource definition. These are constants representing **resource attributes,** which are flags which tell the Resource Manager how to handle the resource. Resource attributes are described by bits in the low-order byte of an integer value:

Bit	*Constant*	*Description*
1	`resChanged`	Resource has been changed.
2	`resPreload`	Resource is to be read into memory immediately after the resource fork is opened.
3	`resProtected`	Application cannot change the resource ID, modify the resource's contents or remove the resource from the resource fork.
4	`resLocked`	Relocatable block occupied by the resource is to be locked. (Overrides the `resPurgeable` attribute.)
5	`resPurgeable`	Relocatable block occupied by the resource is to be purgeable.

Note that, if both the `resPreload` and the `resLocked` attributes are set, the Resource Manager loads the resource as low as possible in the heap.

Resources Which Must Be Unpurgeable. Some resources must *not* be made purgeable. For example, the Menu Manager expects menu resources to remain in memory at all times.

Resources Which May Be Purgeable. Other resources, such as those relating to windows, controls, and dialogs, do not have to remain in memory once the corresponding user interface element has been created. You may therefore set the purgeable attribute for those kinds of resources if you so desire. The following considerations apply to the decision as to whether to make a resource purgeable or unpurgeable:

- The concept of purgeable resources dates back to the time when RAM was limited and programmers had to be very careful about allowing resources which were not in use to continue to occupy precious memory. Nowadays, however, RAM is not so limited.
- Some resources (for example, large `'PICT'` resources and `'snd '` resources) do require a lot of memory, even by today's standards. Accordingly, such resources should generally be made purgeable.
- As will be seen, there are certain hazards associated with the use of purgeable resources. These hazards must be negated by careful programming involving additional lines of code.

Given these considerations, a sound policy would be to make all small and basic resources unpurgeable and set the `resPurgeable` attribute only in the case of comparatively large resources which are not required to remain permanently in memory.

Template Resources and Definition Resources

The `'WIND'` resource defined above is an example of a **template resource.** A template resource defines the characteristics of a desktop object, in this case a window's size, location, and so on, and the **window definition function** (specified by the constant `kWindowDocumentProc`) to be used to draw it. Definition functions, which determine the look and behavior of a desktop object, are executable code segments contained within another kind of resource called a **definition resource.**

Resources in Action

The Resource Map

Your application file's resource fork contains, in addition to the resources you have created for your application, an automatically created **resource map.** The resource map contains entries for each resource in the resource fork.

When your application is launched, the system first gets the Memory Manager to create the application heap and allocate a block of master pointers at the bottom of the heap. The Resource Manager then opens your application file's resource fork and reads in the resource map, followed by those resources which have the `resPreload` attribute set.

The handles to the resources which have been loaded are stored in the resource map in memory. The following is a diagrammatic representation of a simple resource map in memory immediately after the resource map, together with those resources with the preload attribute set, have been loaded.

Type	***ID***		***Attributes***		***Handle***
`MENU`	`128`	**Preload**	**Lock**	**Purgeable**	`123C`
`WIND`	`128`	•		•	`NULL`
`PICT`	`128`			•	`NULL`
`PICT`	`129`			•	`NULL`

Note that the handle entry in the resource map contains `NULL` for those resources that have not yet been loaded. Note also that this handle entry is filled in only when a resource is loaded for the first time, and that that entry remains even if a purgeable resource is later purged by the Memory Manager.

Reading in Non-Preloaded Resources

Some system software managers use the Resource Manager to read in resources for you. Using the `'WIND'` resource listed in the above resource map as an example, when the Window Manager function `GetNewCWindow` is called to create a new window (specifying 128 as the resource ID), `GetNewCWindow`, in turn, calls the Resource Manager function `GetResource`. `GetResource` loads the resource (assuming that it is not currently in memory), returns the handle to `GetNewCWindow`, and copies the handle to the appropriate entry in the resource map. This is an example of an indirect call to the Resource Manager.

Other resources are read in by direct calls to the Resource Manager. For example, the `'PICT'` resources listed in the above example

resource map would be read in by calling another of the `Get…` family of resource-getting functions directly, for example:

```
#define rPicture1 128
#define rPicture2 129
...
PicHandle pic1Hdl;
PicHandle pic2Hdl;
...
pic1Hdl = GetPicture(rPicture1);
pic2Hdl = GetPicture(rPicture2);
```

Once again, and assuming that the resources have not previously been loaded, the handle returned by each `GetPicture` call is copied to the appropriate entry in the resource map.

Purgeable Resources

When a resource that has the `resPurgeable` attribute set has been loaded for the first time, the handle to that resource is copied to the appropriate entry in the resource map in the normal way. If the Memory Manager later purges the resource, the master pointer pointing to that resource is set to `NULL` by the Memory Manager but the handle entry in the resource map remains. This creates what is known as an **empty handle.**

If the application subsequently calls up the resource, the Resource Manager first checks the resource map handle entry to determine whether the resource has ever been loaded (and thus whether a master pointer exists for the resource). If the resource map indicates that the resource has never been loaded, the Resource Manager loads the resource, returns its handle to the calling function, and copies the handle to the resource map.

If, on the other hand, the resource map indicates that the resource has previously been loaded (that is, the handle entry in the resource map contains the address of a master pointer), the Resource Manager checks the master pointer. If the master pointer contains `NULL`, the Resource Manager knows that the resource has been purged, so it reloads the resource and updates the master pointer. Having satisfied itself that the resource is in memory, the Resource Manager returns the resource's handle to the application.

Problems with Purgeable Resources

Using purgeable resources optimises heap space; however, misuse of purgeable resources can crash an application. For example, consider the following code example, which loads two purgeable `'PICT'` resources and then uses the drawing instructions contained in those resources to draw each picture.

```
pic1Hdl = GetPicture(rPicture1);    // Load first 'PICT' resource.
pic2Hdl = GetPicture(rPicture2);    // Load second 'PICT' resource.
if(pic1Hdl)                         // If the handle to first resource is not NULL ...
  DrawPicture(pic1Hdl,&destRect);   // ... draw the first picture.
if(pic2Hdl)                         // If the handle to second resource is not NULL
  DrawPicture(pic2Hdl,&destRect);   // ... draw the second picture.
```

`GetPicture` is one of the many functions that can cause memory to move. When memory is moved, the Memory Manager may purge memory to obtain more heap space. If heap space is extremely limited at the time of the second call to `GetPicture`, the first resource will be purged by the Memory Manager, which will set the master pointer to the first resource to `NULL` to reflect this condition. The variable `pic1Hdl` will now contain an empty handle. Passing an empty handle to `DrawPicture` just about guarantees a crash. (Of course, in the preceding code example, the possibility of a crash is avoided because the line `if(pic1Hdl)` prevents the line from executing.)

There is a second problem with this code. Like `GetPicture`, `DrawPicture` also has the potential to move memory blocks. If the second call to `GetPicture` did not result in the first resource being purged, the possibility remains that it will be purged while it is being used (that is, during the execution of the `DrawPicture` function).

To avoid such problems when using purgeable resources, you should observe these steps:

- Get (that is, load) the resource only when it is needed.
- Immediately make the resource unpurgeable.

- Use the resource immediately after making it unpurgeable.
- Immediately after using the resource, make it purgeable.

The following revised version of the code demonstrates this approach:

```
pic1Hdl = GetPicture(rPicture1);      // Load first 'PICT' resource.
if(pic1Hdl)                           // If the resource was successfully loaded ...
{
  HNoPurge((Handle) pic1Hdl);         // make the resource unpurgeable ...
  DrawPicture(pic1Hdl,&destRect);     // draw the first picture ...
  HPurge((Handle) pic1Hdl);           // and make the resource purgeable again.
}

pic2Hdl = GetResource(rPicture2);     // Repeat for the second 'PICT' resource.
if(pic2Hdl)
{
  HNoPurge((Handle) pic2Hdl );
  DrawPicture(pic2Hdl,&destRect);
  HPurge((Handle) pic2Hdl );
}
```

Note that this procedure only applies when you use functions that get resources directly (for example `GetResource`, `GetPicture`, and so on). It is not required when you call `GetResource` indirectly (for example, when you call the Window Manager function `GetNewCWindow`) because functions like `GetNewCWindow` know how to treat purgeable resources properly.

Note also that `LoadResource` may be used to ensure that a previously loaded, but purgeable, resource is in memory before an attempt is made to use it. If the specified resource is not in memory, `LoadResource` will load it. The essential difference between `LoadResource` and the `Get…` family of resource-getting functions is that the latter *return* a handle to the resource (loading the resource if necessary), whereas `LoadResource` *takes* a handle to a resource as a parameter and loads the resource if necessary.

Releasing Resources

When you have finished using a resource loaded by a function which gets resources directly, you should call the appropriate function to release the memory associated with that resource. For example, `ReleaseResource` is used in the case of generic handles obtained with the `GetResource` function. `ReleaseResource` frees up all the memory occupied by the resource and sets the resource's handle in the resource map to `NULL`.

You do not need to be concerned with explicitly releasing resources loaded indirectly (for example, by a call to `GetNewCWindow`). Using the case of a window resource template as an example, the sequence of events following a call to `GetNewCWindow` is as follows:

- `GetNewCWindow` calls `GetResource` to read in the window resource template whose ID is specified in the `GetNewCWindow` call.
- A relocatable block is created for the template resource and marked as purgeable, as specified by the resource's attributes. (You should always specify window template resources as purgeable.)
- The window template's block is then temporarily marked as unpurgeable while
    - A block is created for an opaque data structure known as a window object.
    - Data is copied from the resource template into the window object.
- The window template's block is then marked as purgeable.

Note

Opaque data structures are explained in Chapter 3.

Resource Manager Errors

The error code resulting from the last call to a Resource Manager function may be retrieved by calling the function `ResError`.

Some of the error codes which may be returned by `ResError` are as follows:

Value	*Constant*	*Description*
`0`	`noErr`	No error occurred.
`-192`	`resNotFound`	Resource not found.
`-193`	`resFNotFound`	Resource file not found.

Main Memory Manager Data Types and Functions

Data Types

```
typedef char *Ptr;     // Pointer to nonrelocatable block.
typedef Ptr  *Handle;  // Handle to relocatable block.
typedef long Size;     // Size of a block in bytes.
```

Function

Allocating and Releasing NonRelocatable Blocks of Memory

```
Ptr     NewPtr(Size byteCount);
Ptr     NewPtrClear(Size byteCount);
void    DisposePtr(Ptr p);
```

Allocating and Releasing Relocatable Blocks of Memory

```
Handle  NewHandle(Size byteCount);
Handle  NewHandleClear(Size byteCount);
Handle  NewEmptyHandle(void);
Handle  NewEmptyHandleSys(void);
void    DisposeHandle(Handle h);
```

Changing the Sizes of Nonrelocatable and Relocatable Blocks

```
Size    GetPtrSize(Ptr p);
void    SetPtrSize(Ptr p,Size newSize);
Size    GetHandleSize(Handle h);
void    SetHandleSize(Handle h,Size newSize);
```

Setting the Properties of Relocatable Blocks

```
void    HLock(Handle h);
void    HUnlock(Handle h);
void    HPurge(Handle h);
void    HNoPurge(Handle h);
SInt8   HGetState(Handle h);
void    HSetState(Handle h,SInt8 flags);
```

Managing Relocatable Blocks

```
void    EmptyHandle(Handle h);
void    ReallocateHandle(Handle h,Size byteCount);
Handle  RecoverHandle(Ptr p);
void    ReserveMem(Size cbNeeded);
void    MoveHHi(Handle h);
void    HLockHi(Handle h);
```

Manipulating Blocks of Memory

```
void    BlockMove(const void *srcPtr,void *destPtr,Size byteCount);
void    BlockMoveData(const void *srcPtr,void *destPtr,Size byteCount);
OSErr   PtrToHand(const void *srcPtr,Handle *dstHndl,long size);
OSErr   PtrToXHand(const void *srcPtr,Handle dstHndl,long size);
OSErr   HandToHand(Handle *theHndl);
OSErr   HandAndHand(Handle hand1,Handle hand2);
```

```
OSErr   PtrAndHand(const void *ptr1,Handle hand2,long size);
```

Allocating Master Pointers

```
void    MoreMasterPointers(UInt32 inCount);
```

Accessing Memory Conditions and Freeing Memory

```
long    FreeMem(void);           // Useful under Mac OS 8/9, almost meaningless under Mac
OS X.
void    PurgeMem(Size cbNeeded); // Useful under Mac OS 8/9, almost meaningless under Mac
OS X.
Size    MaxMem(size *grow);      // Useful under Mac OS 8/9, almost meaningless under Mac
OS X.
long    MaxBlock(void);
void    PurgeSpace(long *total,long *contig);
long    StackSpace(void);        // Not very useful in Carbon.
Size    CompactMem(Size cbNeeded);
```

Checking for Errors

```
OSErr    MemError(void);
```

Main Resource Manager Constants, Data Types, and Functions

Constants

Resource Attributes

```
resSysHeap    = 64  System or application heap?
resPurgeable  = 32  Purgeable resource?
resLocked     = 16  Load it in locked?
resProtected  = 8   Protected?
resPreload    = 4   Load in on OpenResFile?
resChanged    = 2   Resource changed?
```

Data Types

```
typedef unsigned long  FourCharCode;
typedef FourCharCode   ResType;
```

Functions

Reading Resources Into Memory

```
Handle  GetResource(ResType theType,short theID);
Handle  Get1Resource(ResType theType,short theID);
void    LoadResource(Handle theResource);
```

Disposing of Resources

```
void    ReleaseResource(Handle theResource);
```

Checking for Errors

```
short    ResError(void);
```

Demonstration Program SysMemRes Listing

```
//
//****************************************************************************************
// SysMemRes.c
//
//****************************************************************************************
//
// This program:
//
// •  Loads a window template ('WIND') resource and creates a window.
//
// •  Allocates a relocatable block in the application's heap, gets its size in bytes, draws
//    the size in the window, and then disposes of the block.
//
// •  Loads a purgeable 'PICT' resource and a non-purgeable 'STR ' resource and draws them
in
//    the window.
//
// •  Checks if any error codes were generated as a result of calls to Memory Manager and
//    Resource Manager functions.
//
// •  Terminates when the mouse button is clicked.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  A 'WIND' resource (purgeable).
//
// •  A 'PICT' resource (purgeable).
//
// •  A 'STR ' resource (non-purgeable).
//
//
//****************************************************************************************

// ..........................................................................
includes

#include <Carbon.h>

// ..........................................................................
defines

#define rWindowResourceID  128
#define rStringResourceID  128
#define rPictureResourceID 128
// ..................................................................... function
prototypes

void  main            (void);
void  doPreliminaries (void);
void  doNewWindow     (void);
void  doMemory        (void);
void  doResources     (void);

// ****************************************************************************************
main

void  main(void)
{
```

```
  doPreliminaries();
  doNewWindow();
  doMemory();
  doResources();

  QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);

  while(!Button())
    ;
}

// ****************************************************************************
doPreliminaries

void  doPreliminaries(void)
{
  MoreMasterPointers(32);
  InitCursor();
}

// ****************************************************************************
doNewWindow

void  doNewWindow(void)
{
  WindowRef windowRef;

  windowRef = GetNewCWindow(rWindowResourceID,NULL,(WindowRef) -1);
  if(windowRef == NULL)
  {
    SysBeep(10);
    ExitToShell();
  }

  SetPortWindowPort(windowRef);
  UseThemeFont(kThemeSystemFont,smSystemScript);
}
// ****************************************************************************
doMemory

void  doMemory(void)
{
  Handle theHdl;
  Size   blockSize;
  OSErr  memoryError;
  Str255 theString;

  theHdl = NewHandle(1024);
  if(theHdl != NULL)
    blockSize = GetHandleSize(theHdl);

  memoryError = MemError();
  if(memoryError == noErr)
  {
    MoveTo(170,35);
    DrawString("\pBlock Size (Bytes) = ");
    NumToString(blockSize,theString);
    DrawString(theString);

    MoveTo(165,55);
    DrawString("\pNo Memory Manager errors");

    DisposeHandle(theHdl);
  }
}
```

```
// ****************************************************************************
doResources

void  doResources(void)
{
  PicHandle    pictureHdl;
  StringHandle stringHdl;
  Rect         pictureRect;
  OSErr        resourceError;

  pictureHdl = GetPicture(rPictureResourceID);
  if(pictureHdl == NULL)
  {
    SysBeep(10);
    ExitToShell();
  }

  SetRect(&pictureRect,148,75,348,219);

  HNoPurge((Handle) pictureHdl);
  DrawPicture(pictureHdl,&pictureRect);
  HPurge((Handle) pictureHdl);

  stringHdl = GetString(rStringResourceID);
  if(stringHdl == NULL)
  {
    SysBeep(10);
    ExitToShell();
  }

  MoveTo(103,250);
  DrawString(*stringHdl);

  ReleaseResource((Handle) pictureHdl);
  ReleaseResource((Handle) stringHdl);

  resourceError = ResError();
  if(resourceError == noErr)
  {
    MoveTo(160,270);
    DrawString("\pNo Resource Manager errors");
  }
}

//
****************************************************************************************
```

Demonstration Program SysMemRes Comments

The 'plst' (Property List) Resource

As stated in the listing, one of the resources utilized by the program is a 'plst' (property list) resource. Carbon applications must contain a 'plst' resource with ID 0 in order for the Mac OS X Launch Services Manager to recognize them as Carbon applications. If the 'plst' resource is not provided, the application will be launched as a Classic application.

The 'plst' resource used is actually an empty 'plst' resource. As will be seen at Chapter 9, however, this resource, apart from causing Mac OS X to recognize applications as Carbon applications, also allows applications to provide information about themselves to the Mac OS X Finder.

includes

Carbon greatly simplifies the matter of determining which Universal Headers files need to be included in this section in any program. All that is required is to include the header file Carbon.h, which itself causes a great many header files to be included. For this particular program, if Carbon.h is not specified, the following header files would need to be included for the reasons indicated:

```
Appearance.h  Constant:    kThemeSystemFont

              Appearance.h itself includes MacTypes.h, which contains:

            Data Types:  SInt16  Str255  Size  OSErr  Handle  Rect  StringHandle
            Constants:   noErr  NULL
              Appearance.h also includes MacWindows.h, which contains:

              Prototypes:  GetNewCWindow  GetWindowPort

              MacWindows.h itself includes Events.h, which contains:

              Prototype:   Button

              MacWindows.h also includes Menus.h, which itself includes Processes.h, which
contains:

            Prototype:   ExitToShell

              Appearance.h also includes QuickDraw.h, which contains:

              Prototypes:  InitCursor  SetPortWindowPort  SetRect  DrawPicture  MoveTo
                             GetPicture
              Data Types:  WindowRef  PicHandle

              QuickDraw.h itself includes QuickDrawText.h, which contains:

              Prototypes:  TextFont  DrawString

MacMemory.h   Prototypes:  NewHandle  DisposeHandle  GetHandleSize  HNoPurge  HPurge
                             MemError  MoreMasterPointers

Resources.h   Prototypes:  ReleaseResource  ResError

Sound.h       Prototype:   SysBeep

TextUtils.h   Prototype:   GetString

              TextUtils.h itself includes NumberFormatting.h, which contains:

              Prototypes:  NumToString
```

It would be a good idea to peruse the header files MacMemory.h and Resources.h at this stage. Another header file which should be perused early in the piece is MacTypes.h.

defines

Constants are established for the resource IDs of the `'WIND'`, `'PICT'` and `'STR'` resources.

main

The main function calls the functions that perform certain preliminaries common to most applications, create a window, allocate and dispose of a relocatable memory block, and draw a picture and text in the window. It then waits for a button click before terminating the program

The call to the QuickDraw function `QDFlushPortBuffer` is ignored when the program is run on Mac OS 8/9. It is required on Mac OS X because Mac OS X windows are double-buffered. (The picture and text are drawn in a buffer and, in this program, the buffer has to be flushed to the screen by calling `QDFlushPortBuffer`.) This is explained in more detail in Chapter 4.

doPreliminaries

The call to `MoreMasterPointers` is ignored when the program is run on Mac OS X. (Master pointers do not need to be pre-allocated, or their number optimized, in the Mac OS X memory model.)

On Mac OS 8/9, the call to `MoreMasterPointers` to allocate additional master pointers is really not required in this simple program because the system automatically allocates one block of master pointers at application launch. However, in larger applications where more than 64 master pointers are required, the call to `MoreMasterPointers` should be made here so that all master pointer (nonrelocatable) blocks are located at the bottom of the heap. This will assist in preventing heap fragmentation. Note that specifying a number less than 32 in the `inCount` parameter will result in 32 master pointers in the allocated block.

The call to `InitCursor` sets the cursor shape to the standard arrow cursor and sets the cursor level to 0, making it visible.

doNewWindow

`doNewWindow` creates a window.

The call to `GetNewCWindow` creates a window using the `'WIND'` template resource specified in the first parameter. The type, size, location, title, and visibility of the window are all established by the `'WIND'` resource. The third parameter tells the Window Manager to open the window in front of all other windows.

Recall that, as soon as the data from the `'WIND'` template resource is copied to the opaque data structure known as a window object during the creation of the window, the relocatable block occupied by the template will automatically be marked as purgeable.

The call to `SetPortWindowPort` makes the new window's graphics port the current port for drawing operations. The call to the Appearance Manager function `UseThemeFont` sets the font for that port to the system font.

doMemory

`doMemory` allocates a relocatable block of memory, gets the size of that block and draws it in the window (or, more accurately, in the window's graphics port), and checks for memory errors.

The call to `NewHandle` allocates a relocatable block of 1024 bytes. The call to `GetHandleSize` returns the size of the allocated area available for data storage (1024 bytes). (The actual memory used by a relocatable block includes a block header and up to 12 bytes of filler.)

The call to `MemError` returns the error code of the most recent call to the Memory Manager. If no error occurred, the size returned by `GetHandleSize` is drawn in the window, together with a message to the effect that `MemError` returned no error. `DisposeHandle` is then called to free up the memory occupied by the relocatable block.

`MoveTo` moves a "graphics pen" to the specified horizontal and vertical coordinates. `DrawString` draws the specified string at that location. `NumToString` creates a string of decimal digits representing the value of a 32-bit signed integer. These calls are incidental to the demonstration.

doResources

`doResources` draws a picture and some text strings in the window.

`GetPicture` reads in the `'PICT'` resource corresponding to the ID specified in the `GetPicture` call. If the call is not successful, the system alert sound is played and the program terminates.

The `SetRect` call assigns values to the left, top, right and bottom fields of a `Rect` variable. This `Rect` is required for a later call to `DrawPicture`.

The basic rules applying to the use of purgeable resources are to load it, immediately make it unpurgeable, use it immediately, and immediately make it purgeable. Accordingly, the `HNoPurge` call makes the relocatable block occupied by the resource unpurgeable, the `DrawPicture` call draws the picture in the window's graphics port, and the `HPurge` call makes the relocatable block purgeable again. Note that, because `HNoPurge` and `HPurge` expect a parameter of type `Handle`, `pictureHdl` (a variable of type `PicHandle`) must be cast to a variable of type `Handle`.

`GetString` then reads in the specified `'STR '` resource. Once again, if the call is not successful, the system alert sound is played and the program terminates. `MoveTo` moves the graphics "pen" to an appropriate position before `DrawString` draws the string in the window's graphics port. (Since the `'STR '` resource, unlike the `'PICT'` resource, does not have the purgeable bit set, there is no requirement to take the precaution of a call to `HNoPurge` in this case.)

Note the parameter in the call to `DrawString`. `stringHdl`, like any handle, is a pointer to a pointer. It contains the address of a master pointer which, in turn, contains the address of the data. Dereferencing the handle once, therefore, gets the required parameter for `DrawString`, which is a pointer to a string.

The calls to `ReleaseResource` release the `'PICT'` and `'STR '` resources. These calls release the memory occupied by the resources and set the associated handles in the resource map in memory to `NULL`.

The `ResError` call returns the error code of the most recent resource-related operation. If the call returns `noErr` (indicating that no error occurred as a result of the most recent call by a Resource Manager function), some advisory text is drawn in the window.

The next six lines examine the result of the most recent call to a memory manager function and draw some advisory text if no error occurred as a result of that call.

Note that the last two calls to `DrawString` utilize "hard-coded" strings. This sort of thing is discouraged in the Macintosh programming environment. Such strings should ordinarily be stored in a `'STR#'` (string list) resource rather than hard-coded into the source code. The \p token causes the compiler to compile these strings as Pascal strings.

PASCAL STRINGS AND C STRINGS

As stated in the Preface, when it comes to the system software, the ghost of the Pascal language forever haunts the C programmer. For example, a great many system software functions take Pascal strings as a required parameter, and some functions return Pascal strings.

Pascal and C strings differ in their formats. A C string comprises the characters followed by a terminating `0` (or `NULL` byte):



A C string is thus said to be "null-terminated".

In a Pascal string, the first byte contains the length of the string, and the characters follow that byte:



Not surprisingly, then, Pascal strings are often referred to as "length-prefixed" strings.

In the demonstration program, you encountered the data type `Str255`. `Str255` is the C name for a Pascal-style string capable of holding up to 255 characters. As you would expect, the first byte of a `Str255` holds the length of the string and the following bytes hold the characters of the string.

Utilizing 256 bytes for a string will simply waste memory in many cases. Accordingly, the header file MacTypes.h defines the additional types `Str63`, `Str32`, `Str31`, `Str27`, and `Str15`, as well as the `Str255` type:

```
typedef unsigned char Str255[256];
typedef unsigned char Str63[64];
typedef unsigned char Str32[33];
typedef unsigned char Str31[32];
typedef unsigned char Str27[28];
typedef unsigned char Str15[16];
```

Note, then, that a variable of type `Str255` holds the address of an array of 256 elements, each element being one byte long.

As an aside, in some cases you may want to use C strings, and use standard C library functions such as strlen, strcpy, etc., to manipulate them. Accordingly, be aware that functions exist (`CopyPascalStringToC`, `CopyCStringToPascal`) to convert a string from one format to the other.

You may wish to make a "working" copy of the `SysMemRes` demonstration program file package and, using the working copy of the source code file `SysMemRes.c`, replace the function `doResources` with the following, compile-link-run, and note the way the second and third strings appear in the window.

```
void  doResources(void)
{
  Str255  string1 = "\pIs this a Pascal string I see before me?";
  Str255  string2 = "Is this a Pascal string I see before me?";
  Str255  string3 = "%s this a Pascal string I see before me?";
  Str255  string4;
  SInt16  a;

  // Draw string1

  MoveTo(30,100);
  DrawString(string1);

  // Change the length byte of string1 and redraw

  string1[0] = (char) 23;
  MoveTo(30,120);
  DrawString(string1);

  // Leave the \p token out at your peril
  // I (ASCII code 73) is now interpreted as the length byte

  MoveTo(30,140);
  DrawString(string2);

  // More peril:-  % (ASCII code 37) is now the length byte

  MoveTo(30,160);
  DrawString(string3);

  // A hand-built Pascal string

  for(a=1;a<27;a++)
    string4[a] = (char) a + 64;

  string4[0] = (char) 26;
```

```
  MoveTo(30,180);
  DrawString(string4);

  // But is there a Mac OS function to draw the C strings correctly?

  MoveTo(30,200);
  DrawText(string2,0,40);  // Look it up in your reference
}
```

MORE ON STRINGS-UNICODE AND CFString OBJECTS

In the Carbon era, no discussion of strings would be complete without reference to Unicode and `CFString` objects.

Preamble-Writing Systems, Character Encoding and Character Sets,

A writing system (for example, Roman, Hebrew, Arabic) comprises a set of characters and the basic rules for using those characters in creating a visual depiction of a language. An individual character is a symbolic representation of an element of a writing system.

In memory, an individual character is stored as a character code, a numeric value that defines that particular character. One byte (8 bits) is commonly used to store a single character code, which allows for a character set of 256 characters maximum.

A total of 256 numeric values is, of course, quite inadequate to provide for all the characters in all the world's writing systems, meaning that different character sets must be used for different writing systems. In one-byte encoding systems, therefore, these different character sets are said to "overlap."

As an aside, the Apple Standard Roman character set (an extended version of the 128-character ASCII character set) is the fundamental character set for the Macintosh computer. To view the printable characters in this set, replace the main function in the `SysMemRes` demonstration program with the following:

```
void  main(void)
{
  WindowRef windowRef;
  SInt16    fontNum, a,b;
  UInt8     characterCode = 0;

  doPreliminaries();
  windowRef = GetNewCWindow(rWindowResourceID,NULL,(WindowPtr) -1);
  SetPortWindowPort(windowRef);

  GetFNum("\pGeneva",&fontNum);
  TextFont(fontNum);

  for(a = 0;a < 256;a += 16)
  {
    for(b=0;b<16;b++)
    {
      MoveTo((a * 1.5) + 60,(b * 15) + 40);
      DrawText(&characterCode,0,1);
      characterCode++;
    }
  }

  while(!Button())
   ;
}
```

In addition to the overlapping of character sets between writing systems, a further problem is conflicting character encodings within a single writing system. For an example of this in the Roman writing system, change `"\pGeneva"` to

"\pSymbol" in the above substitute main function.

Unicode

Unicode is an international standard that combines all the characters for all commonly used writing systems into a single character set. It uses 16 bits per character, allowing for a character set of up to 65,535 characters. With Unicode, therefore, the character sets of separate writing systems do not overlap. In addition, Unicode eliminates the problem of conflicting character encodings within a single writing system, such as that between the Roman character codes and the codes of the symbols in the Symbol font.

Unicode makes it possible to develop and localize a single version of an application for users who speak most of the world's written languages, including Russian (Cyrillic), Arabic, Chinese, and Japanese.

Core Foundation's String Services

Core Foundation is a component of the system software. Through its String Services, Core Foundation facilitates easy and consistent internationalization of applications. The essential part of this support is an opaque data type (CFString). A CFString "object" represents a string as an array of 16-bit Unicode characters. Several Appearance Manager, Control Manager, Dialog Manager, Window Manager, Menu Manager, and Carbon Printing Manager functions utilize CFStrings.

CFString objects come in immutable and mutable variants. Mutable strings may be manipulated by appending string data to them, inserting and deleting characters, and padding and trimming characters.

CFStrings have many associated functions that do expected things with strings such as comparing, inserting, and appending. Functions are also available to convert Unicode strings (that is, CFStrings) to and from other encodings, particularly 8-bit encodings (such as the Apple Standard Roman character encoding) stored as Pascal and C strings.

Example

For a basic example of the use of CFStrings, remove the calls to doMemory and doResources, and the functions themselves, from the main function in the original version of the demonstration program SysMemRes and replace the function doNewWindow with the following:

```
void  doNewWindow(void)
{
  WindowRef   windowRef;
  Str255      pascalString = "\pThis window title was set using a CFString object";
  CFStringRef titleStringRef;
  CFStringRef textStringRef = CFSTR("A CFString object converted to a Str255");
  Boolean     result;
  Str255      textString;

  windowRef = GetNewCWindow(rWindowResourceID,NULL,(WindowPtr) -1);
  SetPortWindowPort(windowRef);
  UseThemeFont(kThemeSystemFont,smSystemScript);

  titleStringRef = CFStringCreateWithPascalString(NULL,pascalString,
                                                  CFStringGetSystemEncoding());
  SetWindowTitleWithCFString(windowRef,titleStringRef);

  result = CFStringGetPascalString(textStringRef,textString,256,
                                   CFStringGetSystemEncoding());
  if(result)
  {
    MoveTo(135,130);
    DrawString(textString);
  }

  if(titleStringRef != NULL)
    CFRelease(titleStringRef);
}
```

At the sixth line, an immutable `CFString` object is created. The easiest way to do this is to use the `CFSTR` macro. The argument of the macro must be a constant compile-time string (that is, text enclosed in quotation marks) that contains only ASCII characters. `CFSTR` returns a reference (`textStringRef`) to a `CFString` object. This will be used later.

The call to the String Services function `CFStringCreateWithPascalString` converts a Pascal string (`pascalString`, of type `Str255`) to a `CFString` object. The reference to the `CFString` object is then passed in a call to the Window Manager function `SetWindowTitleWithCFString` to set the window's title.

The call to the String Services function `CFStringGetPascalString` converts the `CFString` object (`textStringRef`) previously created by the `CFSTR` macro to a Pascal string (`textString`, of type `Str255`). 256, rather than 255, is passed in the `bufferSize` parameter to accommodate the length byte. `textString` is then passed in a call to the QuickDraw function `DrawString`, which draws the string in the window.

The golden rules for releasing `CFString` objects are if a "Create" or "Copy" function is used, `CFString` should be called to release the string when no longer required; if a "Get" function is used, `CFString` should not be called. Accordingly, `CFRelease` is called only in the case of titleStringRef. Note that `CFRelease` is not `NULL`-safe, so you must check for a non-`NULL` value before passing something to `CFRelease`.

Chapter 2. The Classic Event Manager-Low-Level and Operating System Events

IN THIS CHAPTER

- The Two Event Managers
- Overview of the Classic Event Model
- Obtaining Information About Events
- Flushing the Operating System Event Queue
- Handling Events
- Handling Events in Alerts and Dialogs
- The 'SIZE' Resource
- Main Event Manager Constants, Data Types and Functions
- Demonstration Program LowEvents Listing
- Demonstration Program LowEvents Comments

The Two Event Managers

As stated in Chapter 1, there are two managers in the system software's Human Interface Group that pertain to the subject of events. These two managers are:

- The Event Manager (often unofficially referred to, in the Carbon era, as the Classic Event Manager).
- The somewhat more sophisticated Carbon Event Manager, which was introduced with Carbon.

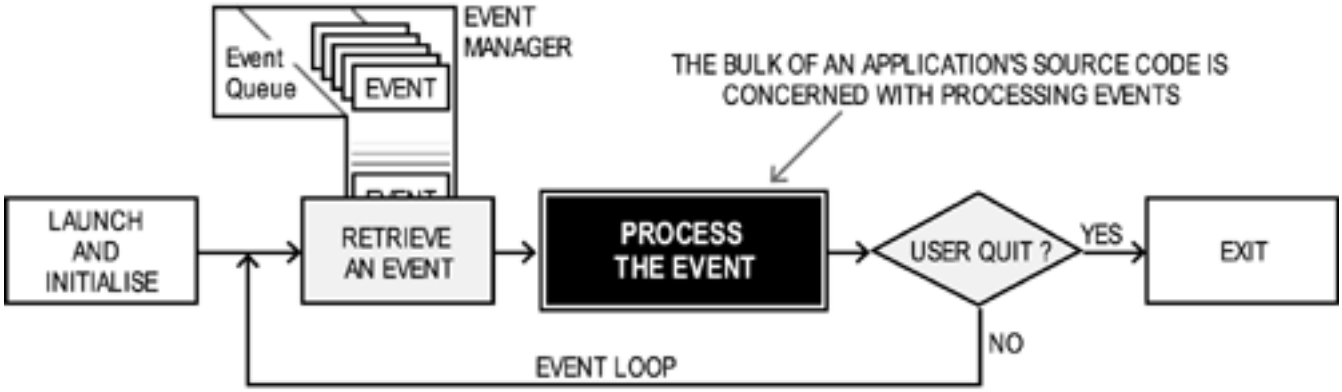
Carbon applications may utilise either of these two event models. This chapter addresses the Classic event model. The Carbon event model is addressed in Chapter 17.

Overview of the Classic Event Model

The Main Event Loop

Any Macintosh application displays one essential characteristic: It is event-driven. At its most basic level, the application's general strategy is to retrieve an **event** (such as a key press or a mouse click), process it, retrieve the next event, process it, and so on indefinitely until the user quits the application. The core of the application is thus the **main event loop** (see Fig 1).

Figure 1. THE MAIN EVENT LOOP



If no events are pending for the active application at a particular time, that application can choose to relinquish control of the CPU (central processing unit, or microprocessor) for a specified amount of time before again checking to see whether an event has occurred. Events are retrieved, and processor time is relinquished, using the `WaitNextEvent` function. The `WaitNextEvent` function is, in a sense, the core of the Classic event model.

Information about a received event is placed in an **event structure.** An application may specify which types of events it wants to receive by including an **event mask** as a parameter in certain Event Manager functions.

Categories of Events

An application can receive many types of events. It can also send certain types of events to other applications. Events are broadly categorized as **low-level** events, **Operating System** events, and **high-level** events. The high-level event is the category of event used to send events to other applications.

Of the three categories, this chapter is concerned only with low-level events and Operating System events. High-level events are addressed in Chapter 10.

Low-Level Events

Low-level events, which are sent to the application by the Toolbox Event Manager, are originated by such low-level occurrences as pressing and releasing a key and pressing and releasing the mouse button.

The Window Manager also originates low-level events, specifically, two events relating to an application's windows:

- The **activate** event, which has to do with informing the application to make changes to the appearance of a window depending on whether or not it is the frontmost window.
- The **update** event, which has to do with informing the application to redraw a window's contents.

The event that reports that the Event Manager has no other events to report (the **null** event) is also categorized as a low-level event.

Low-level events, except for update events and null events, are invariably directed to the foreground process only.

Operating System Events

Operating System events are returned to the application when the operating status of an application changes. For example, when an application has been switched to the background, the Process Manager sends it a **suspend** event. Then, when the application is switched back to the foreground, the Process Manager sends it a **resume** event.

Another Operating System event, called the **mouse-moved** event, is sent when the mouse pointer is moved outside a designated region.

Processes and Events

The subject of **processes** is of some relevance to the subject of events, more particularly to operating system events.

Ordinarily, a user will have more than one application running at the one time. The active application (the application with which the user is currently interacting) is known as the **foreground process.** The remaining open applications, if any, are known as **background processes.** The user can bring a background process to the foreground by, for example, clicking in one of its windows. When an application is switched between background and foreground in this way, a **major switch** is said to have occurred.

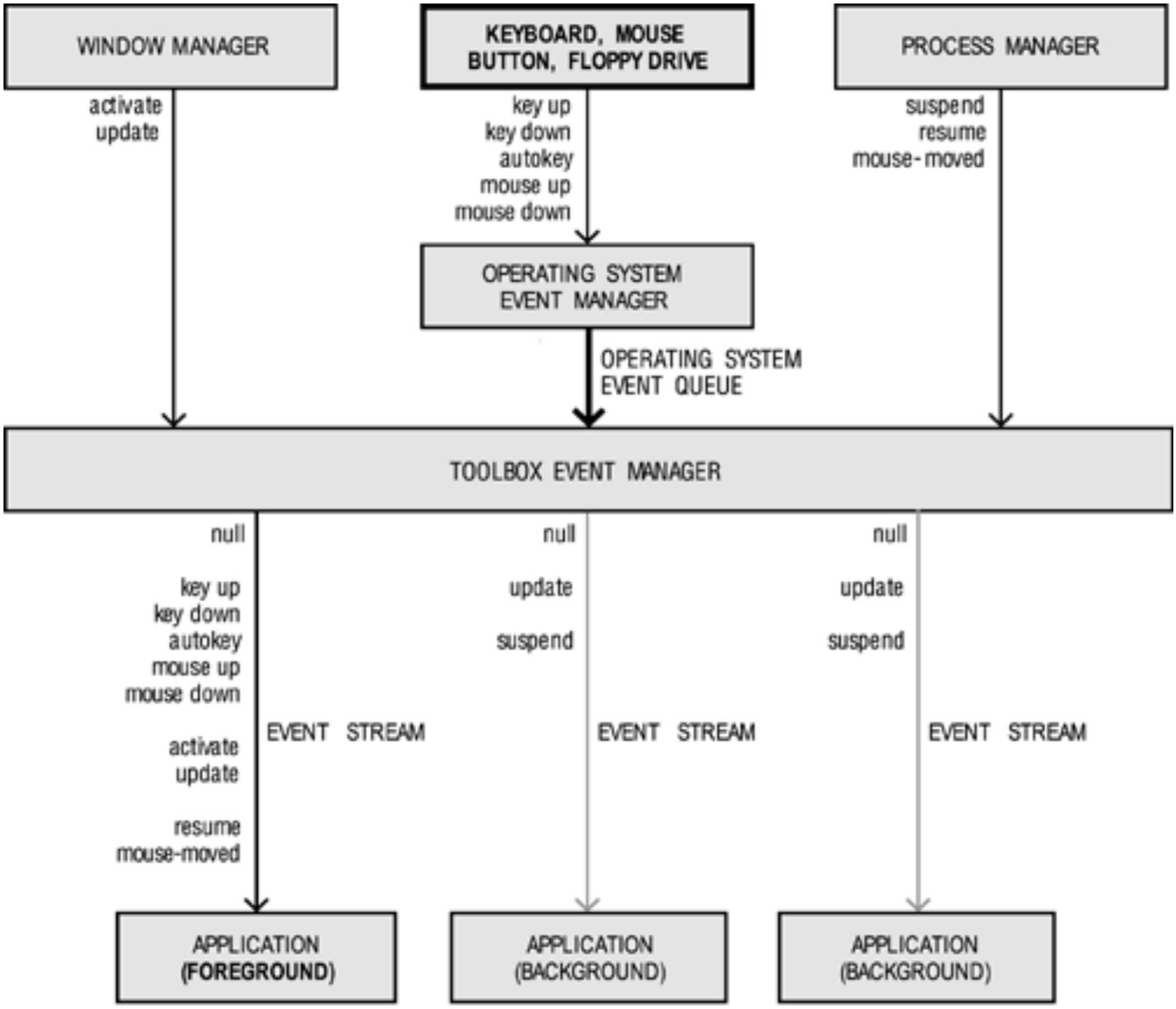
The foreground process has first priority for accessing the CPU, background processes

accessing the CPU only when the foreground process yields time to them. Any application whose `'SIZE'` resource specifies that it should receive null events when it is in the background is eligible for CPU time when it is not in the foreground. A **minor switch** is said to have occurred when a background process gains a period of CPU access without being brought to the foreground.

Low-Level and Operating System Events, System Software, and Applications

Fig 2 shows the relationship between low-level and Operating System events, system software managers and open applications.

Figure 2. LOW-LEVEL AND OPERATING SYSTEM EVENTS



In Fig 2, note that, in addition to the Operating System event queue created by the Operating System Event Manager, the Toolbox Event Manager maintains a separate event stream for each open application. An event stream contains only those events which are available to the related application. Also note that, when an application is in the background, its event stream can

contain only update events, null events, and suspend events, the latter two only if the application's `'SIZE'` resource so specifies.

> **Note**
>
> An application in the background can also receive high-level events. (See Chapter 10.)

A maximum of 48 events can be pending in the Operating System event queue. If the queue becomes full, the oldest event is discarded to make room for the new.

Priority of Events

In general, the Event Manager returns events to the application in the order low-level events, Operating System events, and high-level events. In detail, the order of priority is:

- Activate events
- Mouse-down, mouse-up, key-down, and key-up events in FIFO (first in, first out) order.
- Auto-key events
- Update events, in front-to-back order of windows
- Operating system events
- High-level events
- Null events

Obtaining Information About Events

The Event Structure

The Event Manager continually captures information about each keystroke, mouse click, and so on, and puts information about each event into an **event structure.** As more actions occur, additional event structures are created and joined to the first, forming an event queue.

The `EventRecord` data type defines the event structure:

```
struct EventRecord
{
  EventKind       what;
  UInt32          message;
  UInt32          when;
  Point           where;
  EventModifiers  modifiers;
};

typedef struct EventRecord EventRecord;
```

Field Descriptions

`what` Indicates the type of event received, which may be represented by one of the following constants:

```
nullEvent    = 0   No other pending events.
mouseDown    = 1   Mouse button pressed.
mouseUp      = 2   Mouse button released.
keyDown      = 3   Character key pressed.
keyUp        = 4   Character key released.
autoKey      = 5   Key held down in excess of autoKey threshold.
updateEvt    = 6   Window needs to be redrawn.
activateEvt  = 8   Activate/deactivate window.
osEvt        = 15  Operating system event (suspend, resume or mouse moved).
```

`message` Contains additional information about the event. The content of this field depends on the event type, as follows:

Event Type	*Contents of message Field*

`nullEvent`
`MouseDown`
`mouseUp`
	Undefined.

`keyDown`
`keyUp`
`autoKey`
		Bits 0-7 = character code. Bits 8-15 = virtual key code.
		
		Bits 16-23 = For Apple Desktop Bus keyboards, the ADB address of the keyboard where the event occurred.

`updateEvt` `activateEvt`		Pointer to the window to update, activate or deactivate. (For an `activateEvt`, Bit 0 of the modifiers field indicates whether to activate or deactivate the window.)
`osEvt`	resume	Bits 24-31 = `suspendResumeMessage` constant. Also, a `1` in Bit 0 to indicate that the event is a resume event. Also, a `0` or a `1` in Bit 1 to indicate if clipboard conversion is required.
`osEvt`	suspend	Bits 24-31 = `suspendResumeMessage` constant. Also, a `0` in Bit 0 to indicate that the event is a suspend event.
`osEvt`	mouse-moved	Bits 24-31 = `mouseMovedMessage` constant.

The following constants may be used to extract certain data from, and to test certain bits in, the `message` field:

```
charCodeMask         = 0x000000FF  Mask to extract ASCII character code.
keyCodeMask          = 0x0000FF00  Mask to extract key code.
osEvtMessageMask     = 0xFF000000  Mask to extract OS event message code.
mouseMovedMessage    = 0x00FA      osEvts: mouse-moved event?
suspendResumeMessage = 0x0001      osEvts: suspend/resume event?
resumeFlag           = 1           osEvts: resume event or suspend event?
```

For example, the following code example determines whether an event which has previously been determined to be an Operating System event is a resume event, a suspend event, or a mouse-moved event. In this example, the high byte of the `message` field is examined to determine whether it contains `suspendResumeMessage` (`0x0001`) or `mouseMovedMessage` (`0x00FA`). If it contains `suspendResumeMessage`, Bit 0 is then examined to determine whether the event is a suspend event or a resume event.

```
switch((eventStrucPtr->message >> 24) & 0x000000FF)
{
  case suspendResumeMessage:
    if((eventRecPtr->message & resumeFlag) == 1)
      // This is a resume event.
    else
      // This is a suspend event.
    break;

  case mouseMovedMessage:
    // This is a mouse-moved event.
    break;
}
```

`when` Time the event was posted, in ticks since system startup. (A tick is approximately 1/60th of a second.) Typically, this is used to establish the time between mouse clicks.

`where` Location of cursor, in global coordinates' at the time the event was posted.

Note

Global coordinates are explained at Chapter 4.

`modifiers` Contains information about the state of the modifier keys and the mouse button at the time the event was posted.

For activate events, this field indicates whether the window should be activated or deactivated.

For mouse-down events, this field indicates whether the event caused the application to be switched to the foreground.

Bit	*Description*	
Bit 0	`activateEvt:`	1 if the window pointed to in the message field should be activated. 0 if the window pointed to in the message field should be deactivated.
	`mouseDown:`	1 if the event caused the application to be switched to the foreground, otherwise 0.
Bit 7	1 if mouse button was up, 0 if not.	
Bit 8	1 if Command key down, 0 if not.	
Bit 9	1 if Shiftkey down, 0 if not.	
Bit 10	1 if Caps Lock key down, 0 if not.	
Bit 11	1 if Option key down, 0 if not.	
Bit 12	1 if Control key down, 0 if not.	
Bit 13	1 if Right Shift Key down, 0 if not.	
Bit 14	1 if Right Option Key down, 0 if not.	
Bit 15	1 if Right Control Key down, 0 if not.	

The following constants may be used as masks to test the setting of the various bits in the `modifiers` field:

```
activeFlag      = 0x0001  Window is to be activated? (activateEvt).
                          Foreground switch? (mouseDown).
btnState        = 0x0080  Mouse button up?
cmdKey          = 0x0100  Command key down?
shiftKey        = 0x0200  Shift key down?
AlphaLock       = 0x0400  Caps Lock key down?
optionKey       = 0x0800  Option key down?
controlKey      = 0x1000  Control key down?
rightShiftKey   = 0x2000  Right Shift Key down?
rightOptionKey  = 0x4000  Right Option Key down?
```

```
rightControlKey = 0x8000  Right Control Key down?
```

For example, the following code example determines whether an event which has previously been determined to be an activate event is intended to signal the application to activate or deactivate the window referenced in the `message` field:

```
Boolean becomingActive;

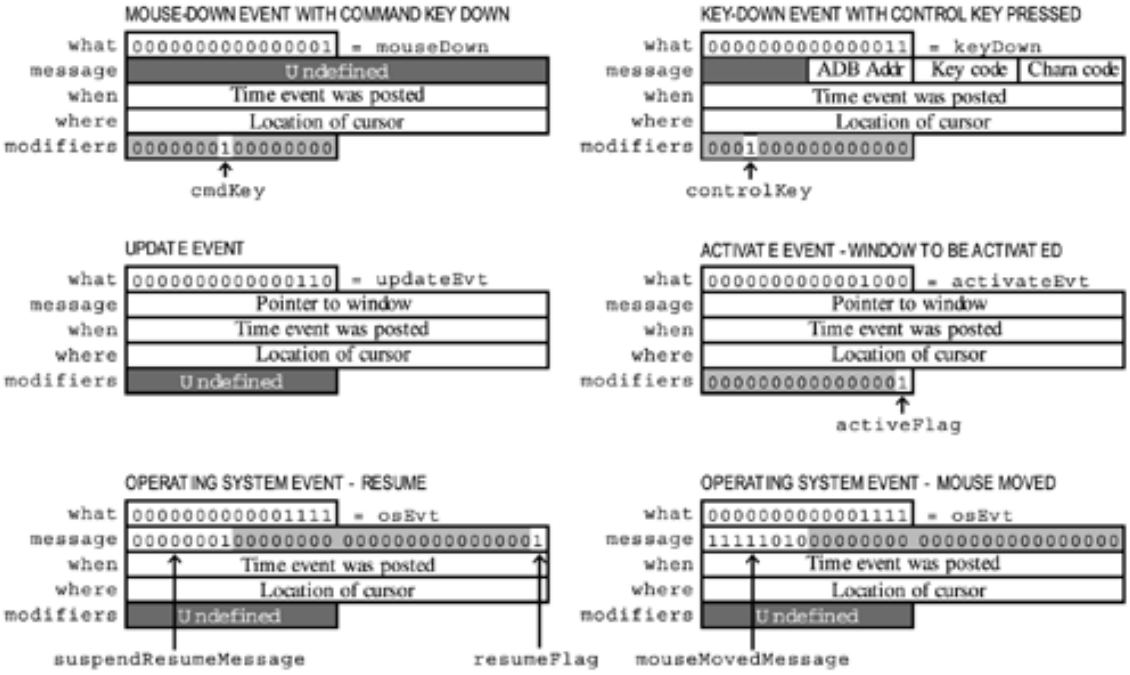
becomingActive = ((eventStrucPtr->modifiers & activeFlag) == activeFlag);

if(becomingActive)
  // Window activation code here.
else
  // Window deactivation code here.
```

Event Structure Examples-Diagrammatic

Fig 3 is a diagrammatic representation of the contents of some typical event structures.

Figure 3. EXAMPLES OF CONTENTS OF AN EVENT RECORD



The WaitNextEvent Function

The `WaitNextEvent` function retrieves events from the Event Manager. If no events are pending for the application, the `WaitNextEvent` function may allocate processor time to other applications. When `WaitNextEvent` returns, the event structure contains information about the retrieved event, if any.

`WaitNextEvent` returns `true` if it retrieves any event other than a null event. If there are no events of the types specified in the `eventMask` parameter (other than null events), `false` is returned.

```
Boolean  WaitNextEvent(EventMask eventMask,EventRecord *theEvent,UInt32 sleep,
                       RgnHandle mouseRgn)
```

Returns: A return code: `0` = null event; `1` = event returned.

`eventMask` A 16-bit binary mask which may be used to mask out the receipt of certain events.

The following constants are defined in Events.h:

```
mDownMask          = 0x0002  Mouse button pressed.
mUpMask            = 0x0004  Mouse button released.
keyDownMask        = 0x0008  Key pressed.
keyUpMask          = 0x0010  Key released.
AutoKeyMask        = 0x0020  Key repeatedly held down.
updateMask         = 0x0040  Window needs updating.
activMask          = 0x0100  Activate/deactivate window.
highLevelEventMask = 0x0400  High-level events (includes AppleEvents).
osMask             = 0x8000  Operating system events (suspend, resume).
everyEvent         = 0xFFFF  All of the above.
```

Masked events are not removed from the event stream by the `WaitNextEvent` call. To remove events from the Operating System event queue, call `FlushEvents` with the appropriate mask.

`theEvent` Address of a 16-byte event structure.

`sleep` On the cooperative multitasking (see below) Mac OS 8/9, the `sleep` parameter specifies the amount of time, in ticks, the application agrees to relinquish the processor if no events are pending for it. If no events are received during this period, `WaitNextEvent` returns `0`, with a null event in the `theEvent` parameter, at the expiration of the sleep period.

On the preemptive multitasking (see below) Mac OS X, the `sleep` parameter is not ignored. It simply causes `WaitNextEvent` to block for the specified period or until an event arrives.

Carbon Note

In order to give drivers time to run, the Classic `WaitNextEvent` will often return long before the sleep time that you pass to it has expired. The Carbon `WaitNextEvent` does not do this; it always waits the full sleep time.

`mouseRgn` The screen **region** inside which the Event Manager does not generate mouse-moved events. The region should be specified in global coordinates. If the user moves the cursor outside this region and the application is the foreground process, the Event Manager reports mouse-moved events.

If `NULL` is passed as this parameter, the Event Manager does not return mouse-moved events.

Before returning to the application, `WaitNextEvent` performs certain additional processing and may, in fact, intercept the received event so that it is never received by your application. As will be seen, key-up and key-down events are intercepted in this way in certain circumstances.

The sleep Parameter and Multitasking

Cooperative Multitasking-Mac OS 8/9

The yielding of access to the CPU by the foreground process, via `WaitNextEvent`'s `sleep` parameter, is central to the form of multitasking provided by the Mac OS 8/9 system software. That form of multitasking is known as **cooperative multitasking.**

Under cooperative multitasking, individual applications continue executing until they "decide" to release control, thus allowing the background process of another application to begin executing. Even though this results in a usable form of multitasking, the operating system itself does not control the processor's scheduling. Even under the best of circumstances, an individual application (which has no way of knowing what other applications are running or whether they have a greater "need" to execute) makes inefficient use of the processor, which often results in the processor idling when it could be used for productive work.

Note also that, under this cooperative scheme, the assignment of zero to `WaitNextEvent`'s `sleep` parameter will cause your application to completely "hog" the CPU whenever it is in the foreground, allowing no CPU time at all to the background processes.

Preemptive Multitasking-Mac OS X

Under **preemptive multitasking,** the operating system itself retains control of which body of code executes, and for how long. No longer does one task have to depend on the good will of another task-that is, the second task's surrender of control-to gain access to the CPU.

Flushing the Operating System Event Queue

Immediately after application launch, the `FlushEvents` function should be called to empty the Operating System event queue of any low-level events left unprocessed by another application-for example, any mouse-down or keyboard events that the user might have entered while the Finder launched the application.

Handling Events

Handling Mouse Events

Your application receives mouse-down events only when it is the foreground process and the user clicks in a window belonging to the application or in the menu bar. (If the user clicks in a window belonging to another application, your application receives a suspend event.)

> **Note**
>
> Events related to the *movement* of the mouse are not stored in the event queue. The mouse driver automatically tracks the mouse and displays the cursor as the user moves the mouse.

The first action on receipt of a mouse-down event is to determine where the cursor was when the mouse button was pressed. A call to `FindWindow` will determine

- Which of your application's windows, if any, the mouse button was pressed in.
- Which window part the mouse button was pressed in. In this context, a window part includes the menu bar as well as various regions within the window.

The following constants, defined in MacWindows.h, may be used to test the value returned by `FindWindow`:

```
inDesk        = 0   In none of the following.
inNoWindow    = 0   In none of the following.
inMenuBar     = 1   In the menu bar.
inContent     = 3   Anywhere in the content region except the grow region if the window
                    is active. Anywhere in the content region including the grow region
                    if the window is inactive.
inDrag        = 4   In the drag region.
inGrow        = 5   In the grow/resize region (active window only).
inGoAway      = 6   In the close region (active window only).
inZoomIn      = 7   In the zoom-in region (active window only).
inZoomOut     = 8   In the zoom-out region (active window only).
inCollapseBox = 11  In the collapse/minimize region (active window only).
inProxyIcon   = 12  In the window proxy icon (active window only).
```

In the Content Region

If the cursor was in the content region of the active window, your application should perform the action that is appropriate to the application. If the window has scroll bars, which actually occupy part of the content region, your application should first determine whether the cursor was in the scroll bars-or, indeed, in any other control-and respond appropriately.

> **Note**
>
> The content region is the part of the window in which an application displays the contents of a document and the window's controls (for example, scroll bars).

In the Title Bar, Size Box, Zoom Box, Close Box, Collapse Box, or Window Proxy Icon

> **Note**

> In the following, Mac OS 8/9 terminology is used. **Size box** equates to **resize control. Zoom box** equates to **zoom button. Close box** equates to **close button. Collapse box** equates to **minimize button.**

If the cursor was in one of the noncontent regions of the active window, your application should perform the appropriate actions for that region as follows:

- ***Title Bar.*** If the cursor was in the title bar, your application should do one of the following:
    - Call `DragWindow` to allow the user to drag the window to a new location. `DragWindow` retains control until the mouse button is released (see Chapter 4).
    - Use an alternative approach introduced with the Mac OS 8.5 Window Manager which first involves a call to IsWindowPathSelect to determine whether the mouse-down event should activate the window path pop-up menu. If IsWindowPathSelect returns true, your application should then call WindowPathSelect to display the menu, otherwise your application should call DragWindow to allow the user to drag the window to a new location (see Chapter 16).
- ***Size Box.*** If the cursor was in the size box, your application should call to `ResizeWindow`, which tracks user actions while the mouse button remains down. When the mouse button is released, `ResizeWindow` draws the window in its new size.
- ***Zoom Box.*** If the cursor was in the zoom box, your application should call to `IsWindowInStandardState` to determine whether the window is currently in the standard state or the user state. `ZoomWindowIdeal` should then be called to zoom the window to the appropriate state, and the window's content region should be redrawn.
- ***Close Box.*** If the cursor was in the close box, your application should call `TrackGoAway` to track user actions while the mouse button remains down. `TrackGoAway`, which returns only when the mouse is released, returns `true` if the cursor is still inside the close box when the mouse button is released, and `false` otherwise.
- ***Collapse Box.*** If the cursor was in the collapse box, your application should do nothing, because the system will collapse (Mac OS 8/9) or minimize (Mac OS X) the window for you.
- ***Window Proxy Icon.*** If the cursor was in the window proxy icon, your application should call `TrackWindowProxyDrag`, which handles all aspects of the drag process while the user drags the proxy icon (see Chapter 16).

In the Menu Bar

If the cursor was in the menu bar, your application should first adjust its menus-that is, enable and disable items and set marks (for, example, checkmarks) based on the context of the active window. It should then call `MenuSelect`, which handles all user action until the mouse button is released.

When the mouse button is released, `MenuSelect` returns a long integer containing, ordinarily, the menu ID in the high word and the chosen menu item in the low word. However, if the cursor was outside the menu when the button was released, the high word contains `0`.

In an Inactive Application Window

If the mouse click was in an inactive application window, `FindWindow` can return only the `inContent` or `inDrag` constant. If `inContent` is reported, your application should bring the inactive window to the front using `SelectWindow`.

Ordinarily, the first click in an inactive window should simply activate the window and do nothing more. However, if the mouse click is in the title bar, for example, you could elect to have your application activate the window *and* allow the user to drag the window to a new location, all on the basis of the first mouse-down.

Detecting Mouse Double Clicks

Double clicks can be detected by comparing the time of a mouse-up event with that of an immediately following mouse-down. `GetDoubleTime` returns the time difference required for two mouse clicks to be interpreted as a double click.

Handling Keyboard Events

After retrieving a key-down event, an application should determine which key was pressed and which modifier keys (if any) were pressed at the same time. Your application should respond appropriately when the user presses a key or combination of keys. For example, your application should allow the user to choose a frequently used menu command by using its keyboard equivalent.

Character Code and Virtual Key Code

The low-order word in the `message` field contains the **character code** and **virtual key code** corresponding to the key pressed by the user.

For a specific key on a particular keyboard, the virtual key code is always the same. The system uses a key-map (`'KMAP'`) resource to determine the virtual key code that corresponds to a specific physical key.

The system software then takes this virtual key code and uses a keyboard layout (`'KCHR'`) resource to map the virtual keycode to a specific character code. Any given script system (that is, writing system) has one or more `'KCHR'` resources (for example, a French `'KCHR'` and a U.S. `'KCHR'`) which determine whether virtual key codes are mapped to, again for example, the French or the U.S. character set.

Generally speaking, your application should use the character code rather than the virtual key code when responding to keyboard events. The following constants may be used as masks to access the virtual key code and character code in the `message` field:

```
keyCodeMask  = 0x0000FF00  Mask to extract key code.
charCodeMask = 0x000000FF  Mask to extract ASCII character code.
```

Checking for Keyboard Equivalents

In its initial handling of key-down and auto-key events, the application should first extract the character code from the `message` field and then check the `modifiers` field to determine if the Command key was pressed at the time of the event. If the Command key was down, the menus should be adjusted prior to further processing of the event. This further processing must necessarily accommodate the possibility that one or more of the modifier keys (Shift, Option, and Control) were also down at the same time as the Command key . If the Command key was not down, the appropriate function should be called to further handle the event.

> **Note**
>
> A menu item can be assigned a keyboard equivalent, that is, any combination of the Command key, optionally one or more modifier keys (Shift, Option, Control), and another key. A Command-key equivalent such as Command-C is also a keyboard equivalent.

Checking For a Command-Period Key Combination

Your application should allow the user to cancel a lengthy operation by using the Command-period combination. This can be implemented by periodically examining the state of the keyboard using `GetKeys` or, alternatively, by calling `CheckEventQueueForUserCancel` to scan the event queue for a Command-period keyboard event. The demonstration program at Chapter 25 contains a demonstration of the latter method.

Events Not Returned to the Application

Certain keyboard events will not, or may not, be returned to your application. These are as follows:

- ***Command-Shift-Numeric Key Combinations.*** Some keystroke combinations are handled by the Event Manager and are

thus not returned to your application. These include certain Command-Shift-numeric key combinations, for example (on Mac OS 8/9), Command-Shift-3 to take a snapshot of the screen. These key combinations invoke a function that takes no parameters and which is stored in an `'FKEY'` resource with a resource ID corresponding to the number key in the Command-Shift-numeric key combination. (Note that IDs of 1 to 4 are reserved by Apple.)

> **Carbon Note**
>
> Function key functions are not supported in Carbon.

- ***Key-Up Events.*** At application launch, the Operating System initializes another event mask, called the **system event mask,** to exclude key-up messages. If an application needs to receive key-up events, the system event mask must be changed using the `SetEventMask` function.

Handling Update Events

Handling Update Events-Mac OS 8/9

The Update Region

On Mac OS 8/9, when one window covers another and the user moves the front window, the Window Manager generates an update event so that the contents of the newly exposed area of the rear window can be updated (redrawn).

The Window Manager keeps track of all areas of a window's content region that need to be redrawn and accumulates them in a region called the **update region.** When the application calls `WaitNextEvent`, the Event Manager determines whether any windows have a non-empty update region. If a non-empty update region is found, the Event Manager reports an update event to the appropriate application. Update events are issued for the front window first when more than one window needs updating,

Updating the Window

Upon receiving the update event, your application should first call `BeginUpdate`, which temporarily replaces the **visible region** of the window's graphics port with the intersection of the visible region and the update region and then clears the update region. This process is explained in more detail in Chapter 4. (If the update region is not cleared, the Event Manager will continue to send an endless stream of update events. Accordingly, it is absolutely essential that `BeginUpdate` be called in response to all update events.)

Your application should then draw the window's contents. (Note that, to prevent the unnecessary drawing of unaffected areas of the window, the system limits redrawing to the visible region, which at this point corresponds to the update region as it was before `BeginUpdate` cleared it.)

`EndUpdate` should then be called to restore the normal visible region.

Update functions should first determine if the window is a document window or a modeless dialog and call separate functions for redrawing the window or the dialog accordingly (see Chapter 8).

Handling Update Events-Mac OS X

On Mac OS X, windows are double-buffered, meaning that your application does not draw into the window's graphics port itself but rather into a separate buffer. The Window Manager flushes the buffer to the window's graphics port when your application calls `WaitNextEvent`. On Mac OS X, your application does not require update events to cater for the situation where part, or all, of a window's content region has just been exposed as a result of the user moving an overlaying window.

On Mac OS X, the receipt of an update event simply means that your application should draw the required contents of the window. The swapping of visible and update regions required on Mac OS 8/9 is not required, so calls to `BeginUpdate` and `EndUpdate` are irrelevant (and ignored) on Mac OS X.

Updating Windows in the Background

Recall that your application will receive update events when it is in the background if the application's `'SIZE'` resource so specifies.

Handling Activate Events

Whenever your application receives a mouse-down event, it should first call `FindWindow` to determine if the user clicked in a window other than the active window. If the click was, in fact, in a window other than the active window, `SelectWindow` should be called to begin the process of activating that window and deactivating the currently active window.

`SelectWindow` does some of the activation/deactivation work for you, such as removing the highlighting from the window being deactivated and highlighting the window being activated. It also generates two activate events so that, at your application's next two requests for an event, an activate event is returned for the window being deactivated followed by an activate event for the window being activated. In response, your application must complete the action begun by `SelectWindow`, performing such actions as are necessary to complete the activation or deactivation process. Such actions might include, for example, showing or hiding the scroll bars, restoring or removing highlighting from any selections, adjusting menus, and so on.

The `message` field of the event structure contains a reference to the window being activated or deactivated and bit 0 of the `modifiers` field indicates whether the window is being activated or deactivated. The `activeFlag` constant may be used to test the state of this bit.

Carbon Note

When the user switches between your application and another application, your application is notified of the switch through Operating System (suspend and resume) events.

In a Classic application, if the application's `'SIZE'` resource has the `acceptSuspendResumeEvents` flag set and the `doesActivateOnFGSwitch` flag not set, your application receives an activate event immediately following all suspend and resume events. This means that the application can rely on the receipt of those activate events to trigger calls to its window activation/deactivation functions when a major switch occurs.

On the other hand, if a Classic application has both the `acceptSuspendResumeEvents` and the `doesActivateOnFGSwitch` flags set, it does not receive an activate event immediately following suspend and resume events. In this case, the application must call its window activation/deactivation functions whenever it receives a suspend or resume event, in addition to the usual call made in response to an activate event.

The generally accepted practice in Classic applications is to set the `doesActivateOnFGSwitch` flag whenever the `acceptSuspendResumeEvents` flag is set.

In a Carbon application, activate events are invariably received along with all suspend and resume events regardless of the `doesActivateOnFGSwitch` flag setting. This would suggest that the `doesActivateOnFGSwitch` flag is irrelevant in a Carbon application and need never be set. However, the correct generation of activate events requires that this flag always be set in a Carbon application.

The upshot is that, in Carbon applications, and despite the fact that the `doesActivateOnFGSwitch` flag should still be set, window activation/deactivation functions need never be called on suspend and resume events.

Handling Null Events

The Event Manager reports a null event when the application requests an event and the application's event stream does not contain any of the requested event types. The `WaitNextEvent` function reports a null event by returning `false` and placing `nullEvt` in the `what` field.

When your application receives a null event, and assuming it is the foreground process, it can perform what is known as **idle processing,** such as blinking the insertion point caret in the active window of the application.

As previously stated, your application's 'SIZE' resource can specify that the application receive null events while it is in the background. If your application receives a null event while it is in the background, it can perform tasks or do other processing.

In order not to deny a reasonable amount of processor time to other applications, idle processing and background processing should generally be kept to a minimum.

Handling Suspend and Resume Events

When an Operating System event is received, the message field of the event structure should be tested with the constants suspendResumeMessage and mouseMovedMessage to determine what type of event was received. If this test reveals that the event was a suspend or resume event, bit 0 should be tested with the constant resumeFlag to ascertain whether the event was a suspend event or a resume event.

WaitNextEvent returns a suspend event when your application has been switched to the background and returns a resume event when your application becomes the foreground process.

> **Carbon Note**
>
> There is a fundamental difference between the receipt of suspend events in Carbon applications as compared with Classic applications. In Classic applications, suspend events are received when your application is *about* to be switched to the background, that is, the application does not actually switch to the background until it makes its next request to receive an event from the Event Manager. In Carbon applications, suspend events are received *after* the application has been switched to the background.

On receipt of a suspend event, your application should do anything necessary to reflect the fact that it is now in the background. When an application receives a resume event, it should do anything necessary to reflect the fact that it is now in the foreground and set the mouse cursor to the arrow shape.

Handling Mouse-Moved Events

Mouse-moved events are used to trigger a change in the appearance of the cursor according to its position in a window. For example, when the user moves the cursor outside the text area of a document window, applications typically change its shape from the I-beam shape to the standard arrow shape.

The main requirement is to specify a region in the mouseRgn parameter of the WaitNextEvent function. This causes the Event Manager to report a mouse-moved event if the user moves the cursor outside that region. On receipt of the mouse-moved event, the application can change the shape of the cursor.

An application might define two regions: a region which encloses the text area of a window (the I-beam region) and a region which defines the scroll bars and all other areas outside the text area (the arrow region). By specifying the I-beam region to WaitNextEvent, the mouse driver continues to display the I-beam cursor until the user moves the cursor out of this region. When the cursor moves outside the region, WaitNextEvent reports a mouse-moved event. Your application can then change the I-beam cursor to the arrow cursor and change the mouseRgn parameter to the non-I-beam region. The cursor now remains an arrow until the user moves the cursor out of this region, at which point your application receives another mouse-moved event.

The application must, of course, recalculate and change the mouseRgn parameter immediately when it receives a mouse-moved event. Otherwise, mouse-moved events will be continually received as long as the cursor is outside the original region.

The appearance of the cursor may be changed using the QuickDraw function SetCursor or the Appearance Manager function SetThemeCursor (see Chapter 6 and Chapter 13).

Cursor setting functions should account for whether a document window or modeless dialog is active and set the cursor appropriately.

Handling Events in Alerts and Dialogs

The handling of events in alerts and dialogs is addressed in detail in Chapter 8. The following is a brief overview only.

Modal and Movable Modal Alerts

The Dialog Manager functions `Alert`, `NoteAlert`, `CautionAlert`, `StopAlert`, and `StandardAlert` are used to invoke modal and movable modal alerts and to handle all user interaction while the alert remains open. The Dialog Manager handles all the events generated by the user until the user clicks a button (typically, the OK or Cancel button). When the user clicks the OK or Cancel button, the Dialog Manager closes the alert and reports the user's action to the application, which is responsible for performing any appropriate subsequent actions.

Modal and Movable Modal Dialogs

For modal and movable modal dialogs, the Dialog Manager function `ModalDialog` is used to handle all user interaction while the dialog is open. When the user selects an item, `ModalDialog` reports the selection to the application, in which case the application is responsible for performing the action associated with that item. An application typically calls `ModalDialog` repeatedly, responding to clicks on enabled items as reported by `ModalDialog`, until the user selects the OK or Cancel button.

Modeless Dialogs

For modeless dialogs, you can use the function `IsDialogEvent` to determine whether the event occurred while a modeless dialog was the frontmost window and then, optionally, use the function `DialogSelect` to handle the event if it belongs to a modeless dialog. `DialogSelect` is similar to `ModalDialog` except that it returns control after every event, not just events relating to an enabled item.

The 'SIZE' Resource

Several references have been made in the preceding to the application's `'SIZE'` resource because some (though not all) of the flag fields in this resource are relevant to the subject of events.

An application's `'SIZE'` resource informs the Operating System

- About the memory requirements of the application
- About certain scheduling options (for example, whether the application can accept suspend and resume events)
- Whether the application
    - Supports stationery documents
    - Supports TextEdit's inline input services
    - Wishes to receive notification of the termination of any application it has launched
    - Wishes to receive high-level events

The `'SIZE'` resource comprises a 16-bit flags field, which specifies the operating characteristics of your application, followed by two 32-bit size fields, one indicating the minimum size, and one the preferred size, of the application's partition.

Resource ID

The `'SIZE'` resource created for your application should have a resource ID of -1. If, on Mac OS 8/9, the user modifies the preferred size in the Finder's Get Info window, the Operating System creates a new `'SIZE'` resource having an ID of 0. If it exists, this latter resource will be invoked by the Operating System at application launch. If it does not exist, the Process Manager looks for the original `'SIZE'` resource with ID -1.

Creating a 'SIZE' Resource in CodeWarrior

It is possible to create a `'SIZE'` resource using Resorcerer; however, it is far more convenient to use the built-in `'SIZE'` resource creation facility within CodeWarrior.

Flags Fields. In CodeWarrior, the bits of the flags field can be set as desired using the 'SIZE' Flags pop-up menu in the PPC Target sections of the Settings dialog, which appears when <Project Name> Settings… is chosen from the Edit Menu. The following describes the meanings of the items in the pop-up menu, and thus of the relevant bits of the 16-bit flags field. Those items relevant to low-level and Operating System events appear on a gray background.

'SIZE' Flags Pop-Up Menu	*Meaning When Set*	*Meaning When Not Set*
`acceptSuspendResumeEvents`	Your application can process, and thus wants to receive suspend and resume events. (When this flag is set, the `doesActivateOnFGSwitch` flag should also normally be set.)	Your application does not want to receive suspend and resume events.
`canBackground`	Your application wants to receive null event processing time while in the background and/ or your application wants to receive suspend events.	Your application does no background processing and thus does not want to receive null events when it is in the background and/or your application does not want to receive suspend events.
`doesActivateOnFGSwitch`	In Classic applications, setting this flag means that your application does not want to receive activate events associated with suspend and resume events, and will thus activate and deactivate its windows in response to suspend and resume events as well as activate events. In a Carbon application, activate events are invariably received along with all suspend and resume events regardless of the `doesActivateOnFGSwitch` flag setting. This would suggest that the `doesActivateOnFGSwitch` flag is irrelevant in a Carbon application and need never be set. However, the correct generation of activate events requires that this flag always be set in a Carbon application.	
`onlyBackground`	Your application runs only in the background. (Usually, this is because it does not have a user interface and cannot run in the foreground.)	Your application runs in the foreground and the background.

`getFrontClicks`	Your application wants to receive the mouse-down and mouse-up events that are used to bring your application into the foreground when the user clicks in your application's frontmost window.	Your application does not want to receive the mouse-down and mouse-up events that are used to bring your application into the foreground.
`acceptAppDiedEvents`	Your application wants to be notified whenever an application launched by your application terminates or crashes. (This information is received via an Apple event.)	Your application does not want to be notified when-ever an application launched by your application terminates or crashes.
`is32BitCompatible`	In pre-Mac OS 8 versions of the system software, setting this flag indicated that your application could be run with either the 32-bit Memory Manager or the 24-bit Memory Manager. Unsetting this flag indicated that your application could not be run with the 32-bit Memory Manager. No Power Macintosh supports the 24-bit mode. Accordingly, this flag is irrelevant in Carbon and should be left unset.	
`isHighLevelEventAware`	Your application can send and receive high-level events. Your application must support the four required Apple events if this flag is set(see Chapter 10).	The Event Manager does not give your application high-level events when it calls `WaitNextEvent`.

For reasons unknown, this flag must always be set in Carbon applications. If this flag is not set, this alert will appear at compile time "Could not launch (application name) because the library ">>CarbonLib<<" could not be found."

`localAndRemoteHLEvents`	Your application is to be visible to applications running on other computers on a network.	Your application does not receive high-level events across a network.

`isStationeryAware`	Your application can recognize stationary documents.	Your application cannot recognize stationary documents. If the user opens a stationary document, the Finder duplicates the document and prompts the user for a name for the duplicate document.
`useTextEditServices`	Your application can use the inline text services provided by TextEdit.	Your application cannot use the inline text services provided by TextEdit.
`isDisplayManagerAware`	Your application can handle the Display Notice event, which tells your application to move its windows after the monitor settings have changed.	When the monitor settings are changed, the DisplayManager moves your application's windows so that they do not disappear off the screen.

Size Fields. For Mac OS 8/9, the minimum and preferred sizes of the application's partition may be set in the Preferred Heap Size (k) and MinimumHeap Size (k) sections of the PPC Target section of the Settings dialog.

Main Event Manager Constants, Data Types and Functions

Constants

Event Codes

```
nullEvent            = 0          No other pending events.
mouseDown            = 1          Mouse button pressed.
mouseUp              = 2          Mouse button released.
keyDown              = 3          Character key pressed.
keyUp                = 4          Character key released.
autoKey              = 5          Key held down in excess of autoKey threshold.
updateEvt            = 6          Window needs to be redrawn.
activateEvt          = 8          Activate/deactivate window.
osEvt                = 15         Operating system event (suspend, resume or mouse moved).
```

Event Masks

```
mDownMask            = 0x0002     Mouse button pressed.
mUpMask              = 0x0004     Mouse button released.
keyDownMask          = 0x0008     Key pressed.
keyUpMask            = 0x0010     Key released.
autoKeyMask          = 0x0020     Key repeatedly held down.
updateMask           = 0x0040     Window needs updating.
activMask            = 0x0100     Activate/deactivate window.
highLevelEventMask   = 0x0400     High-level events (includes AppleEvents).
osMask               = 0x8000     Operating system events (suspend, resume).
everyEvent           = 0xFFFF     All of the above.Event Message
```

Masks for Keyboard Events

```
keyCodeMask          = 0x0000FF00  Mask to extract key code.
charCodeMask         = 0x000000FF  Mask to extract ASCII character code.
```

Message Codes for Operating System Events

```
osEvtMessageMask     = 0xFF000000  Mask to extract OS event message code.
mouseMovedMessage    = 0x00FA      For osEvts, test for mouse-moved event.
suspendResumeMessage = 0x0001      For osEvts, test for suspend/resume event.
resumeFlag           = 1           For osEvts, test Bit 0.
```

Constants Corresponding to Bits in the Modifiers Field

```
activeFlag           = 0x0001      Set if window being activated (activateEvt).
                                   Set if event caused a foreground switch (mouseDown).
btnState             = 0x0080      Set if mouse button up.
cmdKey               = 0x0100      Set if Command key down.
shiftKey             = 0x0200      Set if Shift key down.
alphaLock            = 0x0400      Set if Caps Lock key down.
optionKey            = 0x0800      Set if Option key down.
controlKey           = 0x1000      Set if Control key down.
rightShiftKey        = 0x2000      Set if Right Shift Key down.
rightOptionKey       = 0x4000      Set if Right Option Key down.
rightControlKey      = 0x8000      Set if Right Control Key down.
```

Data Types

Event Structure

```
struct EventRecord
{
  EventKind      what;       // Event code.
  UInt32         message;    // Event message.
  UInt32         when;       // Ticks since system startup.
  Point          where;      // Mouse location in global coordinates.
  EventModifiers modifiers;  // Modifier flags.
}  EventRecord;
typedef struct EventRecord EventRecord;
```

Functions

Receiving Events

```
Boolean  WaitNextEvent(EventMask eventMask,EventRecord *theEvent,UInt32 sleep,
         RgnHandle mouseRgn);
Boolean  EventAvail(EventMask eventMask,EventRecord *theEvent);
void     FlushEvents(EventMask whichMask,EventMask stopMask);
Boolean  GetNextEvent(EventMask eventMask,EventRecord *theEvent)
void     SetEventMask(EventMask value);
```

Reading the Mouse

```
void     GetMouse(Point *mouseLoc);
Boolean  Button(void);
Boolean  StillDown(void);
Boolean  WaitMouseUp(void);
```

Reading the KeyBoard

```
void     GetKeys(KeyMap theKeys);
UInt32   KeyTranslate(const void *transData,UInt16 keycode,UInt32 *state);
```

Getting Timing Information

```
UInt32   TickCount(void);
UInt32   GetDblTime(void);
UInt32   GetCaretTime(void);
```

Demonstration Program LowEvents Listing

```
// ****************************************************************************
// LowEvents.c                                          CLASSIC EVENT MODEL
// ****************************************************************************
//
// This program contains a main event loop function, together with subsidiary
// functions which perform nominal handling only of low-level and Operating
// System events. It opens a window in which the types of all received
// low-level and Operating System events are displayed. It terminates when the
// user clicks the window's close box.
//
// Event handling is only nominal in this program because its main purpose is
// to demonstrate the basics of an application's main event loop. Programs in
// later chapters demonstrate the full gamut of individual event handling.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  A 'WIND' resource (purgeable).
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// ****************************************************************************

// ..........................................................................................................................................................................
includes

#include <Carbon.h>

// ..........................................................................................................................................................................
defines

#define rWindowResource  128

#define topLeft(r)  (((Point *) &(r))[0])
#define botRight(r) (((Point *) &(r))[1])

// ..................................................................................................................................... global variables

Boolean   gDone;
RgnHandle gCursorRegionHdl;

// ..................................................................................................................................... function prototypes

void  main            (void);
void  doPreliminaries (void);
void  doNewWindow     (void);
void  eventLoop       (void);
void  doEvents        (EventRecord *);
void  doMouseDown     (EventRecord *);
void  doUpdate        (EventRecord *);
void  doOSEvent       (EventRecord *);
void  drawEventString (Str255);
void  doAdjustCursor  (WindowRef);

// ********************************************************************* main

void  main(void)
{
  doPreliminaries();
  doNewWindow();
  eventLoop();
}

// ************************************************************ doPreliminaries
```

```
void  doPreliminaries(void)
{
  MoreMasterPointers(48);

  InitCursor();
  FlushEvents(everyEvent,0);
}

// ************************************************************* doNewWindow
void  doNewWindow(void)
{
  WindowRef windowRef;

  if(!(windowRef = GetNewCWindow(rWindowResource,NULL,(WindowRef) -1)))
  {
    SysBeep(10);
    ExitToShell();
  }

  SetPortWindowPort(windowRef);
  TextSize(10);
}

// ************************************************************** eventLoop

void  eventLoop(void)
{
  EventRecord eventStructure;
  Boolean     gotEvent;

  gDone = false;
  gCursorRegionHdl = NewRgn();
  doAdjustCursor(FrontWindow());

  while(!gDone)
  {
    gotEvent = WaitNextEvent(everyEvent,&eventStructure,180,gCursorRegionHdl);
    if(gotEvent)
      doEvents(&eventStructure);
    else
    {
      if(eventStructure.what == nullEvent)
        drawEventString("\p   • nullEvent");
    }
  }
}

// ************************************************************** doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
  switch(eventStrucPtr->what)
  {
    case mouseDown:
      drawEventString("\p   • mouseDown");
      doMouseDown(eventStrucPtr);
      break;

    case mouseUp:
      drawEventString("\p   • mouseUp");
      break;
    case keyDown:
      drawEventString("\p   • keyDown");
      break;

    case autoKey:
      drawEventString("\p   • autoKey");
      break;

    case updateEvt:
      drawEventString("\p   • updateEvt");
      doUpdate(eventStrucPtr);
      break;
```

```
    case activateEvt:
      drawEventString("\p   • activateEvt");
      break;

    case osEvt:
      drawEventString("\p   • osEvt - ");
      doOSEvent(eventStrucPtr);
      break;
  }
}

// ************************************************************* doMouseDown

void  doMouseDown(EventRecord *eventStrucPtr)
{
  WindowPartCode partCode;
  WindowRef      windowRef;

  partCode = FindWindow(eventStrucPtr->where,&windowRef);

  switch(partCode)
  {
    case inContent:
      if(windowRef != FrontWindow())
        SelectWindow(windowRef);
      break;

    case inDrag:
      DragWindow(windowRef,eventStrucPtr->where,NULL);
      doAdjustCursor(windowRef);
      break;

    case inGoAway:
      if(TrackGoAway(windowRef,eventStrucPtr->where))
        gDone = true;
      break;
  }
}
// ************************************************************** doUpdate

void  doUpdate(EventRecord *eventStrucPtr)
{
  BeginUpdate((WindowRef) eventStrucPtr->message);
  EndUpdate((WindowRef) eventStrucPtr->message);
}

// ************************************************************** doOSEvent

void  doOSEvent(EventRecord *eventStrucPtr)
{
  Cursor arrow;

  switch((eventStrucPtr->message >> 24) & 0x000000FF)
  {
    case suspendResumeMessage:
      if((eventStrucPtr->message & resumeFlag) == 1)
      {
        SetCursor(GetQDGlobalsArrow(&arrow));
        DrawString("\pResume");
      }
      else
        DrawString("\pSuspend");
      break;

    case mouseMovedMessage:
      doAdjustCursor(FrontWindow());
      DrawString("\pMouse-moved");
      break;
  }
}

// ********************************************************* drawEventString
```

```
void  drawEventString(Str255 eventString)
{
  WindowRef windowRef;
  RgnHandle tempRegion;
  Rect      portRect;

  windowRef = FrontWindow();
  tempRegion = NewRgn();

  GetWindowPortBounds(windowRef,&portRect);
  ScrollRect(&portRect,0,-15,tempRegion);
  DisposeRgn(tempRegion);

  MoveTo(8,340);
  DrawString(eventString);
}
// ************************************************************  doAdjustCursor

void  doAdjustCursor(WindowRef windowRef)
{
  RgnHandle myArrowRegion;
  RgnHandle myIBeamRegion;
  Rect      cursorRect;
  Point     mousePt;
  Cursor    arrow;

  myArrowRegion = NewRgn();
  myIBeamRegion = NewRgn();
  SetRectRgn(myArrowRegion,-32768,-32768,32767,32767);

  GetWindowPortBounds(windowRef,&cursorRect);
  SetPortWindowPort(windowRef);
  LocalToGlobal(&topLeft(cursorRect));
  LocalToGlobal(&botRight(cursorRect));

  RectRgn(myIBeamRegion,&cursorRect);
  DiffRgn(myArrowRegion,myIBeamRegion,myArrowRegion);

  GetGlobalMouse(&mousePt);
  if(PtInRgn(mousePt,myIBeamRegion))
  {
    SetCursor(*(GetCursor(iBeamCursor)));
    CopyRgn(myIBeamRegion,gCursorRegionHdl);
  }
  else
  {
    SetCursor(GetQDGlobalsArrow(&arrow));
    CopyRgn(myArrowRegion,gCursorRegionHdl);
  }

  DisposeRgn(myArrowRegion);
  DisposeRgn(myIBeamRegion);
}

// ********************************************************************************
```

Demonstration Program LowEvents Comments

When the program is run, the user should move the mouse cursor inside and outside the window, click the mouse inside and outside the window, drag the window, and press and release keyboard keys, noting the types of events generated by these actions as printed on the scrolling display inside the window.

The user should also note

- The basic window deactivation and activation that occurs when the mouse is clicked outside, and then inside the window.
- That when another window is positioned over part of the program's window and then dragged to expose more of the program's window, an update event is received on Mac OS 8/9 but not on Mac OS X.

The program may be terminated by a click in the window's close box.

defines

`rWindowResource` establishes a constant for the ID of the `'WIND'` resource.

The remaining two lines define two common macros. The first converts the top and left fields of a `Rect` to a Point. The second converts the bottom and right field of a `Rect` to a Point.

Global Variables

The global variable `gDone` controls the termination of the main event loop and thus of the program. `gCursorRegionHdl` will be assigned the handle to a region to be passed in the mouseRgn parameter of the `WaitNextEvent` function.

main

The main function calls the functions for performing certain preliminary actions common to most applications and for creating the window. It then calls the function containing the main event loop.

doPreliminaries

`doPreliminaries` is the standard "do preliminaries" function which will be used in all subsequent Classic event model demonstration programs.

`FlushEvents` empties the Operating System event queue of any low-level events left unprocessed by another application, for example, any mouse-down or keyboard events that the user may have entered while this program was being launched.

doNewWindow

The function `doNewWindow` opens the window in which the types of low-level and Operating System events will be printed as they occur. The `'WIND'` resource passed as the first parameter specifies that the window has a close box and a drag (title) bar. The window's graphics port is set as the current port for drawing and the text size is set to 10 points.

eventLoop

`eventLoop` is the main event loop.

The global variable `gDone` is set to false before the event loop is entered. This variable will be set to true when the user clicks on the window's close box. The event loop (the while loop) terminates when `gDone` is set to true.

The calls to `NewRgn` and `doAdjustCursor` have to do with the generation of mouse-moved events. The `NewRgn` call allocates storage for a Region object and initializes the contents of the region to make it an empty region. As will be seen, this first call to `doAdjustCursor` defines two regions (one for the arrow cursor and one for the I-Beam cursor) and copies the handle to one of them (depending on the current position of the mouse cursor) to the global variable `gCursorRegionHandle`.

In the call to `WaitNextEvent`

- The event mask `everyEvent` ensures that all types of low-level and Operating System events will be returned to the application (except `keyUp` events, which are masked out by the system event mask).
- `eventStructure` is the `EventRecord` structure which, when `WaitNextEvent` returns, will contain information about the event.
- 180 represents the number of ticks for which the application agrees to relinquish the processor if no events are pending for it. 180 ticks equates to about three seconds.
- If the cursor is now not within the region passed in the `cursorRegion` parameter, a mouse-moved event will be generated immediately.

`WaitNextEvent` returns `1` if an event was pending, otherwise it returns `0`. If an event was pending, the program branches to `doEvent` to determine the type of event and handle the event according to its type. If `0` is returned, and if the what field of the event structure contains `nullEvent`, `"nullEvent"` is printed in the window. This will occur every three seconds in the absence of other events.

doEvents

`doEvents` handles some events to finality and performs initial handling of others.

On return from `WaitNextEvent`, the what field of the event structure contains an unsigned short integer that indicates the type of event received. The `doEvent` function isolates the type of event and switches according to that type.

In this demonstration, the action taken in every case is to print the type of event in the window. In addition, and in the case of mouse-down, update, and Operating System events only, calls to individual event handling functions are made.

Note that, in the case of an Operating System event, `doEvent` will only print `"osEvt - "` in the window. At this stage, the program has not yet established whether the event is a suspend, resume or mouse-moved event.

Note also that the inclusion of the key-up event handling would be pointless, since key-up events are masked out by the Operating System.

doMouseDown

The function `doMouseDown` handles mouse-down events to completion.

`FindWindow` is called to get a reference to the window in which the event occurred and a part code which indicates the part of that window in which the mouse-down occurred. The function then switches according to that part code.

The `inContent` case deals with a mouse-down in a window's content region. `FrontWindow` returns a reference to the frontmost window. If this is not the same as the reference in the event structure's message field, `SelectWindow` is called to generate activate events and to perform basic window activation and deactivation. (Actually, `SelectWindow` will never be called in this demonstration because the program only opens one window, which is always the front window.)

The `inDrag` case deals with a mouse-down in the window's title bar (Mac OS 8/9) or title bar (Mac OS X). In this case, control is handed over to `DragWindow`, which tracks the mouse and drags the window according to mouse movement until the mouse button is released. A bounding rectangle limiting the area in which the window can be dragged may be passed in `DragWindow`'s third parameter. In Carbon, `NULL` may also be passed in this parameter. This has the effect of setting the third parameter to the bounding box of the "desktop region" (also known as the "gray region"). The desktop region is the region below the menu bar, including all screen real estate in a system equipped with multiple monitors.

The regions controlling the generation of mouse-moved events are defined in global coordinates. The region for the I-Beam cursor is based on the window's graphics port's bounding rectangle. Accordingly, when the window is moved, the new location of the port rectangle, in global coordinates, must be recalculated so that the arrow cursor and I-Beam cursor regions may be redefined. The call to `doAdjustCursor` redefines these regions for the new window location and copies the handle to one of them, depending on the current location of the mouse cursor, to the global variable `gCursorRegionHandle`. (Note that this call to `doAdjustCursor` will also be required, for the same reason, when a window is resized or zoomed.)

The `inGoAway` case deals with the case of a mouse-down in the close box (Mac OS 8/9) or close button (Mac OS X). In this case, control is handed over to `TrackGoAway`, which tracks the mouse while the mouse button remains down. When the button is released,

`TrackGoAway` returns true if the cursor is still inside the close box, in which case the global variable `gDone` is set to true, terminating the event loop and the program.

doUpdate

The function `doUpdate` handles update events to completion.

Although no window updating is performed by this program, it is nonetheless necessary to call `BeginUpdate` because `BeginUpdate` clears the update region, thus preventing the generation of an unending stream of update events. The call to `EndUpdate` always concludes a call to `BeginUpdate`, undoing the results of the visible/update region manipulations of the latter.

doOSEvent

`doOSEvent` first determines whether the Operating System event passed to it is a suspend/resume event or a mouse-moved event by examining bits 24-31 of the message field. It then switches according to that determination.

In the case of a suspend/resume event, a further examination of the message field establishes whether the event was a suspend event or a resume event. In the case of a resume event, the call to `SetCursor` ensures that the cursor will be set to the arrow cursor shape when the application comes to the foreground. (With regard to the call to `GetQDGlobalsArrow`, see the QuickDraw Globals and Accessor Functions sidebar.)

In the case of a mouse-moved event (which occurs when the mouse cursor has moved outside the region whose handle is currently being passed in `WaitNextEvent`'s `mouseRgn` parameter), `doAdjustCursor` is called to change the handle passed in the `mouseRgn` parameter according to the current location of the mouse.

drawEventString

`drawEventString` is incidental to the demonstration. It simply prints text in the window indicating when the various types of events are received. `ScrollRect` scrolls the contents of the current graphics port within the rectangle specified in the first parameter. The second parameter specifies the number of pixels to be scrolled to the right and the third parameter specifies the number of pixels to scroll vertically, in this case 15 up.

doAdjustCursor

`doAdjustCursor`'s primary purpose in this particular demonstration is to force the generation of mouse-moved events. The fact that it also changes the cursor shape simply reflects the fact that changing the cursor shape is usually the sole reason for generating mouse-moved events in the first place.

Basically, the function establishes two regions (the calls to `NewRgn`), one describing the content area of the window (in global coordinates) and the other everything outside that. The location of the cursor, in global coordinates, is then ascertained by the call to `GetGlobalMouse`. If the cursor is in the content area of the window (the I-Beam region), the cursor is set to the I-Beam shape and the handle to the I-Beam region is copied to the global variable passed in the `mouseRgn` parameter in the `WaitNextEvent` call in the `eventLoop` function. If the cursor is in the other region (the arrow region), the cursor is set to the normal arrow shape and the arrow region is copied to the global variable passed in the `mouseRgn` parameter.

`GetCursor` reads in the system `'CURS'` resource specified by the constant `iBeamCursor` and returns a handle to the 68-byte Cursor structure created by the call. The parameter for a `SetCursor` call is required to be the address of a Cursor structure. Dereferencing the handle once provides that address.

`WaitNextEvent`, of course, returns a mouse-moved event only when the cursor moves outside the "current" region, the handle to which is passed in the `mouseRgn` parameter of the `WaitNextEvent` call. Only one mouse-moved event, rather than a stream of mouse-moved events, will be generated when the cursor is moved outside the "current" region because

- The mouse-moved event will cause `doAdjustCursor` to be called
- `doAdjustCursor` will thus reset the "current" region to the region in which the cursor is now located

The cursor and cursor adjustment aspects, as opposed to the region-swapping aspects, of the `doAdjustCursor` function are incidental to the demonstration. These aspects are addressed in more detail in Chapter 13.

QUICKDRAW GLOBALS AND ACCESSOR FUNCTIONS

An accessor function (`GetQDGlobalsArrow`) pertaining to application QuickDraw global variables is used in this demonstration.

QuickDraw global variables are stored as part of your application's global variables. In Carbon, accessor functions are provided, and must be used, to access the data in these globals. The accessor functions are as follows:

```
CGrafPtr GetQDGlobalsThePort(void);              // Gets pointer to current graphics port.
Cursor*  GetQDGlobalsArrow(Cursor *arrow);       // Gets standard cursor arrow shape.
void     SetQDGlobalsArrow(const Cursor *arrow); // Sets standard cursor arrow shape.
Pattern* GetQDGlobalsDarkGray(Pattern *dkGray);  // Gets pre-defined dark gray pattern.
Pattern* GetQDGlobalsLightGray(Pattern *ltGray); // Gets pre-defined light gray pattern.
Pattern* GetQDGlobalsGray(Pattern *gray);        // Gets pre-defined gray pattern.
Pattern* GetQDGlobalsBlack(Pattern *black);      // Gets pre-defined black pattern.
Pattern* GetQDGlobalsWhite(Pattern *white);      // Gets pre-defined white pattern.
long     GetQDGlobalsRandomSeed(void);           // Get random number generator seed.
void     SetQDGlobalsRandomSeed(long randomSeed);    // Set random number generator seed.
BitMap*  GetQDGlobalsScreenBits(BitMap *screenBits); // screenBits.bounds contains
                                                     // rectangle enclosing main screen.
```

Chapter 3. Menus

IN THIS CHAPTER

- Introduction-Types of Menus
- Pull-Down Menus
- Hierarchical Menus
- Pop-Up Menus
- Menu Objects, Menu IDs and Item Numbers, Command IDs, and Menu Lists
- Creating Your Application's Menus
- Providing Help Balloons (Mac OS 8/9)
- Changing Menu Item Appearance
- Adding Items to a Menu
- Associating Data with Menu Items
- Handling Menu Choices
- Hiding and Showing the Menu Bar
- **Accessing Menus From Alerts and Dialogs**
- Main Menu Manager Constants, Data Types, and Functions
- Demonstration Program Menus1 Listing
- Demonstration Program Menus1 Comments
- Demonstration Program Menus2 Listing
- Demonstration Program Menus2 Comments

Introduction-Types of Menus

A menu is a user interface element that allows the user to view, or choose from, a list of choices and commands provided by your application. There are basically three types of menus:

- ***Pull-Down Menus.*** A pull-down menu comprises a menu title, displayed in the menu bar, and one or more menu items.

- ***Submenus.*** A submenu is a menu that is attached to another menu. A menu to which a submenu is attached is referred to as a **hierarchical menu.**

- ***Pop-Up Menus.*** A pop-up menu is a menu that does not appear in the menu bar but rather appears on another part of the screen.

Pull-Down Menus

Menu Definition Functions and Menu Bar Definition Functions

The Menu Manager uses the following to display, and to perform basic operations on, menus and the menu bar:

- ***Menu Definition Function.*** When you define a menu, you must specify the required menu definition function (MDEF). The Menu Manager uses that MDEF to draw the menu items in a menu, determine which item the user chose, and so on. An MDEF thus determines the look and behavior of menus.

- ***Menu Bar Definition Function.*** The Menu Manager uses the menu bar definition function (MBDF) to draw and clear the menu bar, determine whether the cursor is currently within the menu bar or any currently displayed menu, highlight menu titles, and so on. A menu bar definition function thus determines the look and behavior of the menu bar.

Standard Menu and Menu Bar Definition Functions

The system software provides a standard MDEF and a standard MBDF. The standard MDEF is the `'MDEF'` resource with a resource ID of 63. The standard MBDF is the `'MBDF'` resource with a resource ID of 63.

Ordinarily, your application will specify the standard definition functions; however, as with most other elements of the Macintosh user interface, the option is available to write your own custom definition function if you need to provide features not available in the standard definition functions.

The Menu Bar and Menus

The Menu Bar

The menu bar extends across the top of the screen and is high enough to display menu titles in the height of the large system font (Mac OS 8/9) or system font (Mac OS X).

Generally, the menu bar should always be visible. If you want to hide the menu bar for some reason, you should provide a method (for example, a keyboard equivalent) to allow the user to make the menu bar reappear.

The 'MBAR' Resource

Each application has its own menu bar, which is defined by an `'MBAR'` resource. This resource lists the order and resource ID of each menu appearing in your menu bar. Your application's `'MBAR'` resource should be defined such that the Mac OS 8/9 Apple menu or Mac OS X Application menu (see below) is the first menu in the menu bar, with the File menu being the next. For Mac OS 8/9, the Help menu and the Mac OS 8/9 Application menu do not need to be defined in the `'MBAR'` resource, because the Menu Manager automatically adds them to the menu bar when the application calls `GetNewMBar`, provided that your menu bar includes the Apple menu.

Menus

All Macintosh applications should ordinarily provide the Mac OS 8/9 Apple menu or Mac OS X Application menu, a File menu, and a Window menu (see Chapter 16). If your application is not document-oriented, the File menu may be renamed to something more appropriate.

Your application can disable any menu, which causes the Menu Manager to dim that menu's title and all associated menu items. The menu items can also be disabled individually. Your application should specify whether menu items are enabled or disabled when it first defines and creates a menu and can enable or disable items at any time thereafter.

The 'MENU' Resource

For each menu, you define the menu title and the individual characteristics of its menu items in a `'MENU'` resource.

The 'xmnu' Resource

For each menu, you may also define an `'xmnu'` (extended menu) resource. The `'xmnu'` resource is, in effect, an extension of the `'MENU'` resource required to provide for additional menu features. . Note that you do not need to provide this resource if you do not require these additional features. An `'xmnu'` resource must have the same ID as the `'MENU'` resource it extends.

Menu Items

A menu item can contain text or a **divider.** On Mac OS 8/9 the divider is a line extending the full width of the menu. On Mac OS X it is simply an empty space, like a menu item with no text. Each menu item, other than a divider, can have a number of characteristics as follows:

- An icon, small icon, reduced icon, color icon, or an icon from an icon family to the left of the menu item's text.

Note

The various icon types are described in Chapter 13.

- A checkmark or other **marking character** indicating the status of the menu item or the mode it controls.
- The symbols for the item's keyboard equivalent. (An item that has a keyboard equivalent cannot have a submenu, a small icon or a reduced icon.)
- A triangular indicator to the right of a menu item's text to indicate that the item has a submenu. (An item that has a submenu cannot have a keyboard equivalent, a small icon or a reduced icon.)
- A font style (bold, italic, and so on) for the menu item's text.
- The text of the menu item.
- The ellipsis character (…) as the last character in the text of the menu item, indicating that the application will display a dialog requesting more information from the user before executing the command. (The ellipsis character should not be used in menu items that display informational dialogs or a confirmational alert.)
- A dimmed appearance when the application disables the item. (When the menu title is dimmed, all menu items in that menu are also dimmed.)

Groups of Menu Items

Where appropriate, menu items should be grouped, with each group separated by a divider. For example, a menu can contain commands that perform actions and commands that set attributes. The action commands that are logically related should be grouped, as should attribute commands that are interdependent. The attribute commands that are mutually exclusive, and those that form accumulating attributes (for example, Bold, Italic and Underline), should also be grouped.

Keyboard Equivalents for Menu Commands

The Menu Manager provides support for **keyboard equivalents.** Your application can detect a keyboard equivalent by examining the `modifiers` field of the event structure, first determining whether the Command key was pressed at the time of the event. If a keyboard equivalent is detected, your application typically calls `MenuEvent`, which maps the keyboard equivalent character contained in the specified event structure to its corresponding menu and menu item and returns the menu ID and the chosen menu item.

Note

A **keyboard equivalent** is any combination of the Command key, optionally one or more modifier keys (Shift, Option, Control), and another key. A **Command-**

key equivalent such as Command-C is thus, by definition, also a keyboard equivalent.

Reserved Command-Key Equivalents

Apple reserves the following Command-key equivalents, which should be used in the File and Edit menus of your application:

Keys	***Command***	***Menu***
Command-A	Select All	Edit
Command-C	Copy	Edit
Command-N	New	File
Command-H	Hide <appname>	Application (Mac OS X)
Command-M	Minimize Window	Window (Mac OS X)
Command-O	Open…	File
Command-P	Print…	File
Command-Q	Quit	File (Mac OS 8/9)
		Application (Mac OS X)
Command-S	Save	File
Command-V	Paste	Edit
Command-W	Close	File
Command-X	Cut	Edit
Command-Z	Undo	Edit

Other common keyboard equivalents are

Keys	***Command***	***Menu***
Command-B	Bold	Style
Command-F	Find	File

Command-G	Find Again	File
Command-I	Italic	Style
Command-T	Plain Text	Style
Command-U	Underline	Style

The Mac OS 8/9 Apple Menu and Mac OS X Application Menu

On Mac OS 8/9, the Mac OS 8/9 Apple Menu is the first menu in your application. On Mac OS X, the Mac OS X Application Menu (see Fig 1) is the first menu.

Figure 1. MAC OS X APPLICATION MENU



Typically, applications provide an About command as the first menu item in the Apple (Mac OS 8/9) and Mac OS X Application menus. On Mac OS 8/9, the remaining items are controlled by the contents of the Apple Menu Items folder in the System folder. On Mac OS X, the remaining items are the default items automatically included in the system-created Mac OS X Application menu. Mac OS X Application menu items are general to the application, that is, they are items that are not specific to a document or other window.

To create your application's Mac OS 8/9 Apple menu for Mac OS 8/9, you simply define the Apple menu title and the characteristics of your application's About command in a `'MENU'` resource. When your application is run on Mac OS 8/9, the contents of the Apple Menu Items folder are automatically added to the Apple menu.

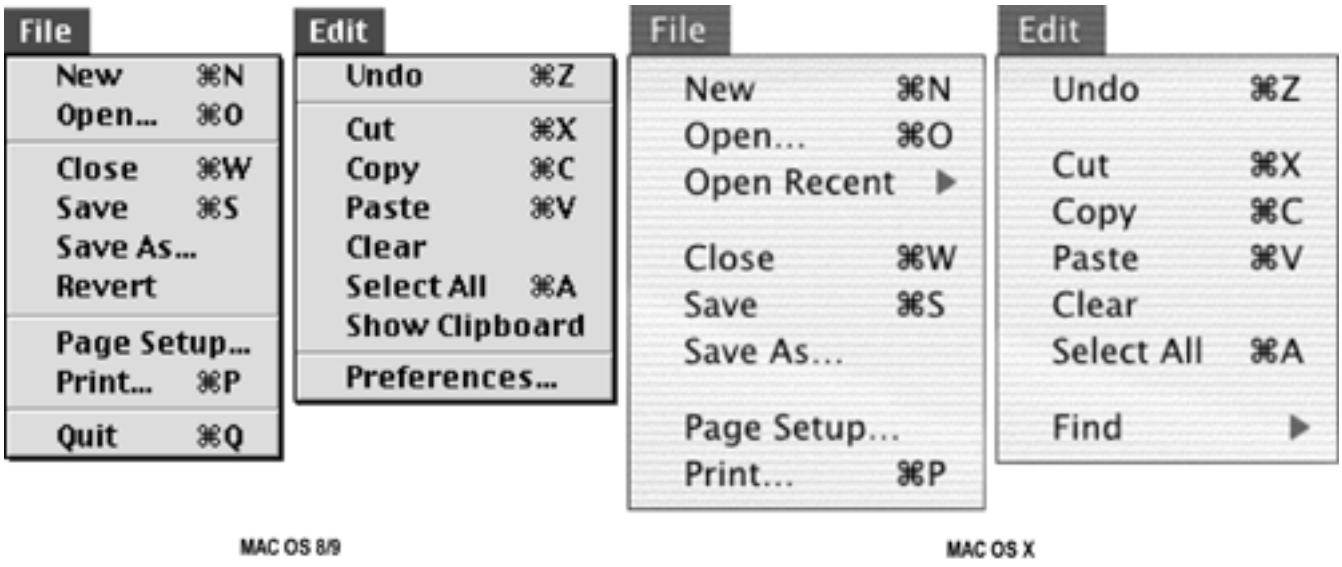
The Apple menu `'MENU'` resource will also cause the About command to be inserted in the Mac OS X Application menu when the application is run on Mac OS X.

When the user chooses the About command on Mac OS 8/9, your application should display a dialog or an alert containing your application's name, version number, copyright information, any other information as required. On Mac OS X, your application should display a modeless dialog containing the application's version and copyright information, as prescribed in Aqua Human Interface Guidelines.

The File Menu

The standard File menu contains commands related to document management plus, on Mac OS 8/9, the Quit command. (On Mac OS X, the Quit command is located in the Application menu (refer to Fig 1).) The standard commands should be supported by your application where appropriate (see Fig 2). Any other commands added to the menu should pertain to the management of documents. The actions your application should take when File menu commands are chosen are detailed in Chapter 15 and Chapter 18.

Figure 2. THE STANDARD FILE AND EDIT MENUS



The Edit Menu

The standard Edit menu provides commands related to the editing of a document's contents, to copying data between different applications using the Clipboard and, on Mac OS 8/9, to showing and hiding the Clipboard (refer to Fig 2). For Mac OS 8/9 only, the standard Edit

menu also standardizes the location of the Preferences… command, which, when chosen, should invoke a dialog that enables the user to set application-specific preferences. (On Mac OS X, the Preferences… command is located in the Mac OS X Application menu.)

> **Note**
>
> The implementation of Preferences commands is demonstrated at the demonstration program in Chapter 19.

All Macintosh applications which support text entry, including text entry in edit text items in dialogs, should include the standard editing commands (Undo, Cut, Copy, Paste and Clear). An additional word or phrase should be added to Undo to clarify exactly what action your application will reverse.

Other commands may be added if they are related to editing or changing the contents of your application's documents.

The Mac OS 8/9 System-Managed Menus

On Mac OS 8/9, two menus, namely the Mac OS 8/9 Application menu and the Help menu, are added automatically by the Menu Manager and are often referred to as the **system- managed menus.**

The Mac OS 8/9 Application Menu

When the user chooses an item from the Mac OS 8/9 Application menu, the Menu Manager handles the event as appropriate. For example, if the user chooses another application, your application is sent to the background and receives a suspend event.

The Mac OS 8/9 Help Menu

Applications written for Mac OS 8/9 using the Classic API have the option of programmatically appending an item (or items) to the end of the Help menu, and of programmatically detecting the user's choice of that item, so as to give the user access to help texts provided by the application. This option is not available in the Carbon API.

Carbon applications may use **Apple Help,** which was introduced with Mac OS 8.6, to provide application help. Apple Help documentation and tools are included in an Apple Help Software Development Kit (SDK), which is available at http://developer.apple.com/sdk. The documentation describes how to create an Apple Guide file, which, when located in the same folder as your application, will cause the system to install a help menu item (or items) in the Help menu. The menu at the left in Fig 3 show the Help menu as it normally appears. The

menus at the right at Fig 3 show the Help menu as it appears when the Apple Guide file is present.

Figure 3. THE MAC OS 8/9 HELP MENU-EFFECT OF THE APPLE GUIDE FILE



Mac OS Help Menus

For Mac OS X, your application must itself create the Help menu (using the function `HMGetHelpMenu`), insert the required item, or items, in that menu, and respond to the user choosing items in the menu.

Font Menus

If your application has a Font menu, you should list in that menu the names of all currently available fonts (that is, all those residing in the Fonts folder in the System folder). Fonts may be added to the Font menu using `AppendResMenu` or `InsertResMenu`. However, a better alternative is to use the relatively new Menu Manager function `CreateStandardFontMenu` to build either a hierarchical or non-hierarchical Font menu. (A hierarchical Font menu is one in which the styles available in each font family (regular, bold, italic, and so on) appear in a submenu attached to the menu item containing the font family name.)

To indicate which font is currently in use in a non-hierachical Font menu, your application should add a checkmark to the left of the font's name in the Font menu. If the current selection contains more than one font, a dash should be placed next to the name of each font the selection contains. When the user starts entering text at the insertion point, your application should display text in the current font.

To indicate which font is currently in use in a hierachical Font menu, your application should place a checkmark next to the font in the submenu and a dash next to the menu item to which the submenu is attached.

Font Attributes

Separate menus should be used to accommodate lists of font attributes such as styles and sizes.

WYSIWYG Font Menus

The function `SetMenuItemFontID` allows you to easily set up a Font menu with each item being drawn in the actual font.

Hierarchical Menus

A hierarchical menu is a menu which has a submenu attached to it. You should use a submenu only when you have more menus than fit in the menu bar. There should only ever be one hierarchical level, that is, there should be only one level of submenus. A menu item that is the title of a submenu should clearly represent the choices the submenu contains.

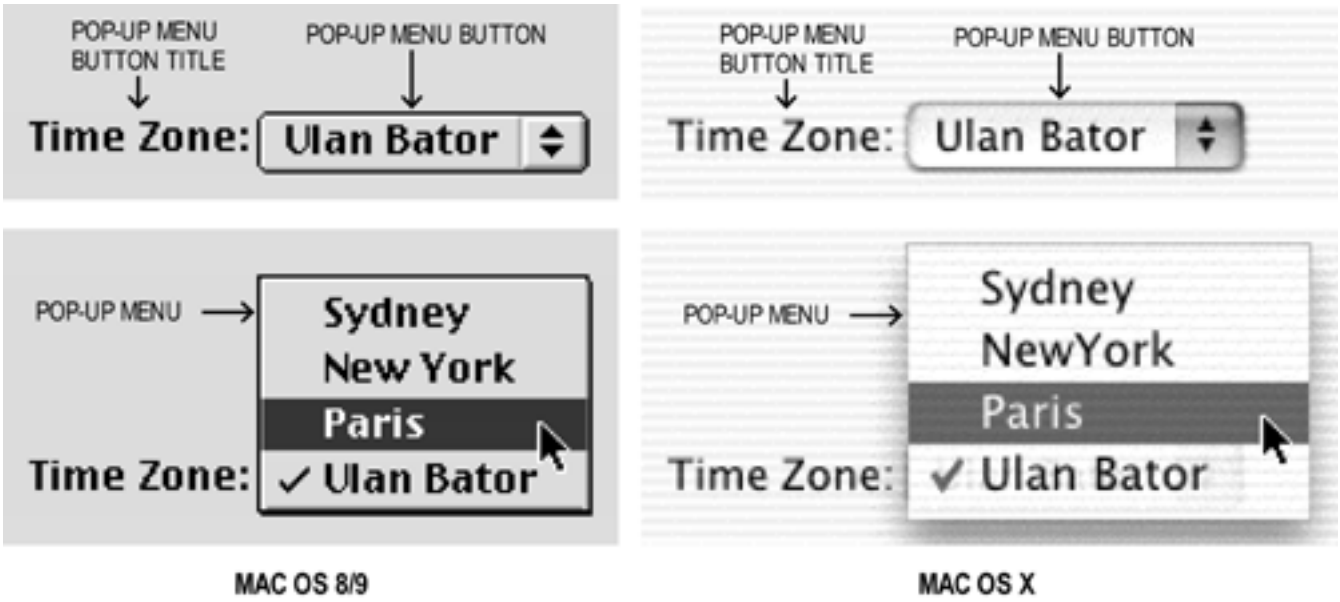
Hierarchical menus work best for providing a submenu of attributes.

Pop-Up Menus

Pop-up menus work well when your application needs to present several choices to the user and it is acceptable to hide these choices until the menu is opened. (Other methods of displaying choices are checkboxes and radio buttons.) Pop-up menus should not be used for multiple choice lists or as a way to provide more commands. They should contain attributes rather than actions; accordingly, Command-key equivalents should not be used in pop-up menus.

The standard pop-up menu is actually implemented as a **control,** specifically, **the pop-up menu button control.** Its appearance and behavior is thus determined by a pop-up menu button control definition function (see Fig 4).

Figure 4. POP-UP MENU BUTTON (EXAMPLE)



Because pop-up menus are implemented as controls, they are addressed at Chapter 7. Further information in this chapter will be limited to the provision of the `'MENU'` resource required by the pop-up menu button control.

Menu Objects, Menu IDs and Item Numbers, Command IDs, and Menu Lists

The Menu Object

The Menu Manager maintains information about individual menus in opaque data structures known as **menu objects.** The data type `MenuHandle` is defined as a pointer to a menu object:

```
typedef struct OpaqueMenuHandle* MenuHandle;
```

Note that the data type `MenuHandle` is equivalent to the newer data type `MenuRef`:

```
typedef MenuHandle MenuRef;
```

Carbon Note

A major change introduced in Carbon is that some commonly used data structures are now **opaque,** meaning that their internal structure is hidden to applications. Directly referencing fields within these structures is no longer possible, and special new **accessor functions** must be used instead.

As an example, the Classic API equivalent of the menu object is the `MenuInfo` structure, which is defined as follows:

```
struct MenuInfo
{
  MenuID  menuID;
  short   menuWidth;
  short   menuHeight;
  Handle  menuProc;
  long    enableFlags;
  Str255  menuData;
} ;
typedef struct MenuInfo MenuInfo;
typedef MenuInfo *MenuPtr;
typedef MenuPtr *MenuHandle;
```

In the Classic API, your application can determine the menu width by directly accessing the `menuWidth` field like this:

```
MenuHandle menuHdl;
SInt16     width;

menuHdl = GetMenuHandle(mFile);  // Get handle to MenuInfo structure.
width   = (**menuHdl).menuWidth;
```

In Carbon, you must use the accessor function `GetMenuWidth` to obtain the menu width from a menu object:

```
MenuRef menuRef;
SInt16  width;

menuRef = GetMenuRef(mFile);    // Get reference to menu object.
width   = GetMenuWidth(menuRef);
```

The following accessor functions are provided to access the information in menu objects:

Function	*Description*
`GetMenuID`	Gets the menu ID of the specified menu.
`SetMenuID`	Sets the menu ID of the specified menu.
`GetMenuWidth`	Gets the horizontal dimensions, in pixels, of the specified menu.
`SetMenuWidth`	Sets the horizontal dimensions, in pixels, of the specified menu.
`GetMenuHeight`	Gets the vertical dimensions, in pixels, of the specified menu.
`SetMenuHeight`	Sets the vertical dimensions, in pixels, of the specified menu.
`GetMenuTitle`	Gets the title of the specified menu.
`SetMenuTitle`	Sets the title of the specified menu.
`SetMenuTitleWithCFString`	
`GetMenuDefinition`	Gets a pointer to a custom menu definition function that has already been associated with the menu. (There is no way to get a pointer to the system menu definition function.)
`SetMenuDefinition`	Sends a dispose message to the current menu definition and an init message to the new definition.

You typically specify most of the information in a menu object in a `'MENU'` resource. When you create a menu, the Menu Manager creates a menu object for the menu and returns a reference to that object. The Menu Manager automatically updates the menu object when you make any changes to the menu programmatically.

Menu IDs and Item Numbers

To refer to a menu, you usually use either the menu's ID or the reference to the menu's menu object. Accordingly, you must assign a **menu ID** to each menu in your application as follows:

- Pull-down menus must use a menu ID greater than 0.

- Submenus of an application may use a menu ID in the range 1 to 32767.

To refer to a menu item, you use the item's **item number.** Item numbers in a menu start at 1.

Command IDs

The **command ID,** a unique value that you set to identify a menu item, is an alternative way of referring to a specific menu item in an application's menus.

The Menu List

The **menu list,** a structure private to the Menu Manager, contains references to the menu objects of one or more menus (although a menu list can, in fact, be empty). The end of a menu list contains references to the menu objects of submenus and pop-up menus, if any, the phrase "submenu portion of the menu list" referring to this portion of the list.

At application launch, the Menu Manager creates the menu list. The menu list is initially empty but changes as your application adds menus to it or removes menus from it programmatically.

Creating Your Application's Menus

'MBAR', 'MENU', and 'xmnu' Resources

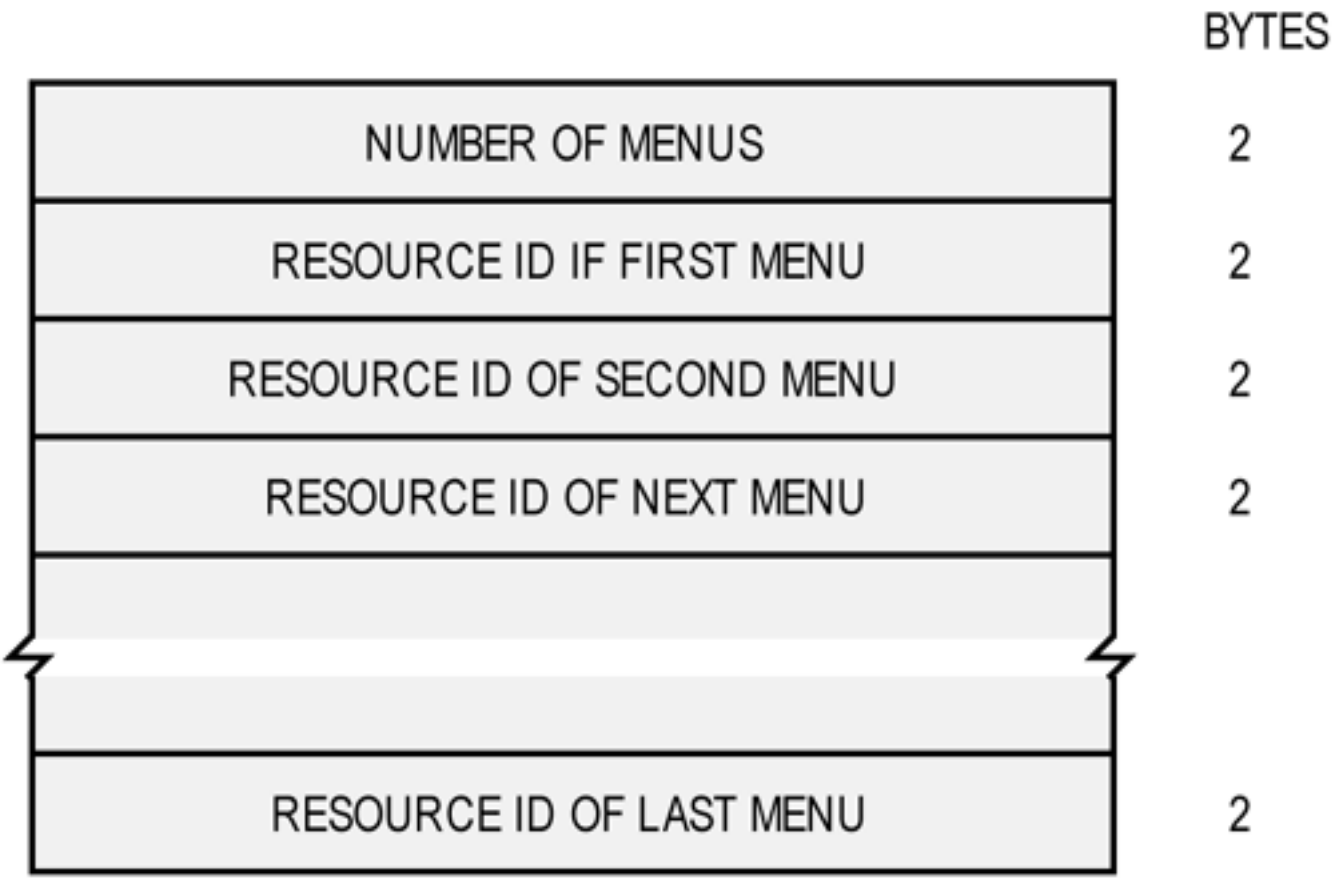
As stated in Chapter 1, you can provide a textual, formal description of resources in a file and then use a resource compiler such as Rez to compile the description into a resource, or you can create resource descriptions using a resource editor such as Resorcerer. This book assumes the use of Resorcerer.

When creating resources using Resorcerer, it is advisable that you refer to a diagram and description of the structure of the resource and relate that to the various items in the Resorcerer editing windows. Accordingly, the following describes the structure of those resources associated with the creation of menus.

Structure of a Compiled 'MBAR' Resource

Fig 5 shows the structure of a compiled `'MBAR'` resource. The number of menu resource IDs should match the number of menus declared in the first two bytes.

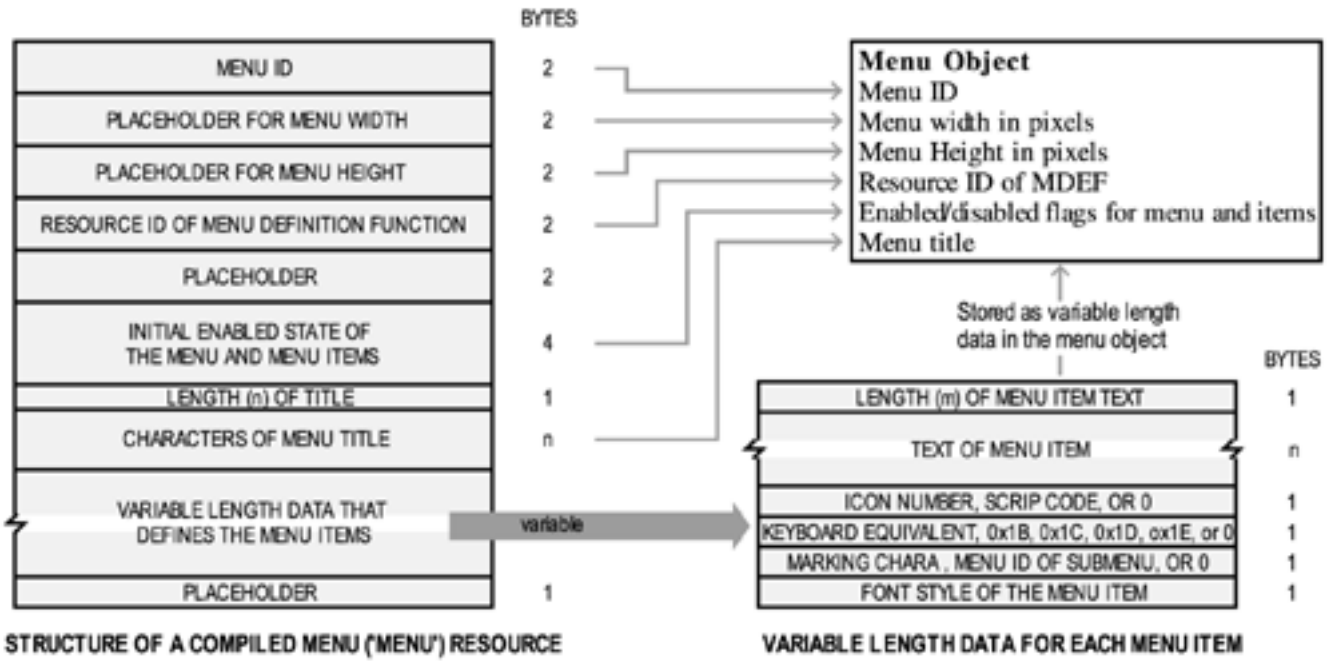
Figure 5. STRUCTURE OF A COMPILED 'MBAR' RESOURCE



Structure of a Compiled 'MENU' Resource

Fig 6 shows the structure of a compiled `'MENU'` resource (and its variable length data) and how it "feeds" the menu object.

Figure 6. STRUCTURE OF A COMPILED MENU ('MENU') RESOURCE AND ITS VARIABLE LENGTH DATA



The following describes the main fields of the `'MENU'` resource:

Field	***Description***
MENU ID	The menu's unique identification number. Note that the number assigned to the menu ID and the resource ID do not have to be identical, though it is advisable that these numbers be the same. A menu ID from 1 to 235 indicates a menu (or submenu) of an application. Apple reserves the menu ID of 0.
PLACEHOLDER FOR MENU WIDTH PLACEHOLDER FOR MENU HEIGHT	After reading in the resource data, the Menu Manager requests the menu's MDEF to calculate the width and height of the menu and store these values in the menu object.
RESOURCE ID OF MENU DEFINITION FUNCTION	If the integer 63 appears here, the standard MDEF will be used. The MDEF is read in after the menu's resource data is read in. The Menu Manager stores a handle to the MDEF in the menu object.

INITIAL ENABLED STATE OF THE MENU AND MENU ITEMS	A value whose bits indicate if the corre sponding menu item is enabled or disabled, with bit 0 indicating whether the menu as a whole is enabled or disabled.
VARIABLE LENGTH DATA THAT DEFINES THE MENU ITEMS	The Menu Manager stores the variable length data for each menu item at the end of the menu object (see Fig 6).

The following describes the main fields of the variable length data for each menu item. Note that various alternatives apply to the icon number, keyboard equivalent, and marking character fields. For example, a menu item can have a keyboard equivalent or a submenu, but not both.

Field	***Description***
ICON NUMBER, SCRIPTCODE, OR 0	ICON NUMBER A number from 1 to 255 (or from 1 to 254 for small or reduced icons). If the menu item specifies an icon, you must provide a `'cicn'` (color icon) or `'ICON'` resource with a resource ID equal to the icon number plus 256. If you want an `'ICON'` resource to be reduced to the size of a small icon (`'SICN'`), or if you want the size of a `'cicn'` resource reduced by half, assign the value `0x1D` to the keyboard equivalent field. If you want a `'SICN'` resource, assign the value `0x1E`. The Menu Manager looks first for a `'cicn'` resource with the calculated resource ID. In the Carbon era, color icons are much to be preferred. SCRIPTCODE (Not used when the 'MENU' resource is extended with an 'xmnu' resource) Specify the script code here if you want the item's text to be drawn in a script other than the system script, and also provide `0x1C` in the keyboard equivalent field.

When the `'MENU'` resource is extended by an `'xmnu'` resource, the script code should be set in the text encoding field of the `'xmnu'` resource.

0

Specifies that the menu item does not contain an icon and uses the system script.

KEYBOARD EQUIVALENT, 0X1B, 0X1C,

KEYBOARD EQUIVALENT 0X1D, OX1E, OR 0

Specified as a one-byte character and, actually, a Command-key equivalent only.

The Command-key equivalent can be extended with modifier key (Shift, Option, Control) constants in the modifier keys field of the extended menu (`'xmnu'`) resource (see below).

0x1B

Specifies that the menu item has a submenu. (The menu ID of the submenu should be assigned to the marking character field (see below).)

0x1C (Not used when the 'MENU' resource is extended with an 'xmnu' resource)

Specifies that the item uses a script other than the system script. (The script code should be assigned to the icon number field (see above).)

When the `'MENU'` resource is extended by an `'xmnu'` resource, the script code should be set in the text encoding field of the `'xmnu'` resource.

0x1D

	For menu items containing icons, causes `'ICON'` resources to be reduced to the size of a small icon, or `'cicn'` resources to be reduced by half. 0x1E Specifies that you want the Menu Manager to use a small icon (`'SICN'`) resource for the item's icon. (The small icon's resource ID should be assigned to the icon number field (see above).) 0 Specifies that the menu item has neither a keyboard equivalent nor a submenu and uses the system script.
MARKING CHARACTER, MENU ID OF	MARKING CHARACTER SUBMENU, OR 0 Special characters, such as the checkmark and diamond characters are available to indicate the marks associated with a menu item. MENU ID OF SUBMENU Submenus of an application must have menu IDs from 1 to 235. Submenus of a driver must have menu IDs from 236 to 255. 0 Specifies that the item has neither a mark nor a submenu.
FONT STYLE OF THE MENU ITEM	Specifies whether the font style of the menu item should be plain, or any combination of bold, italic, outline, and shadow.

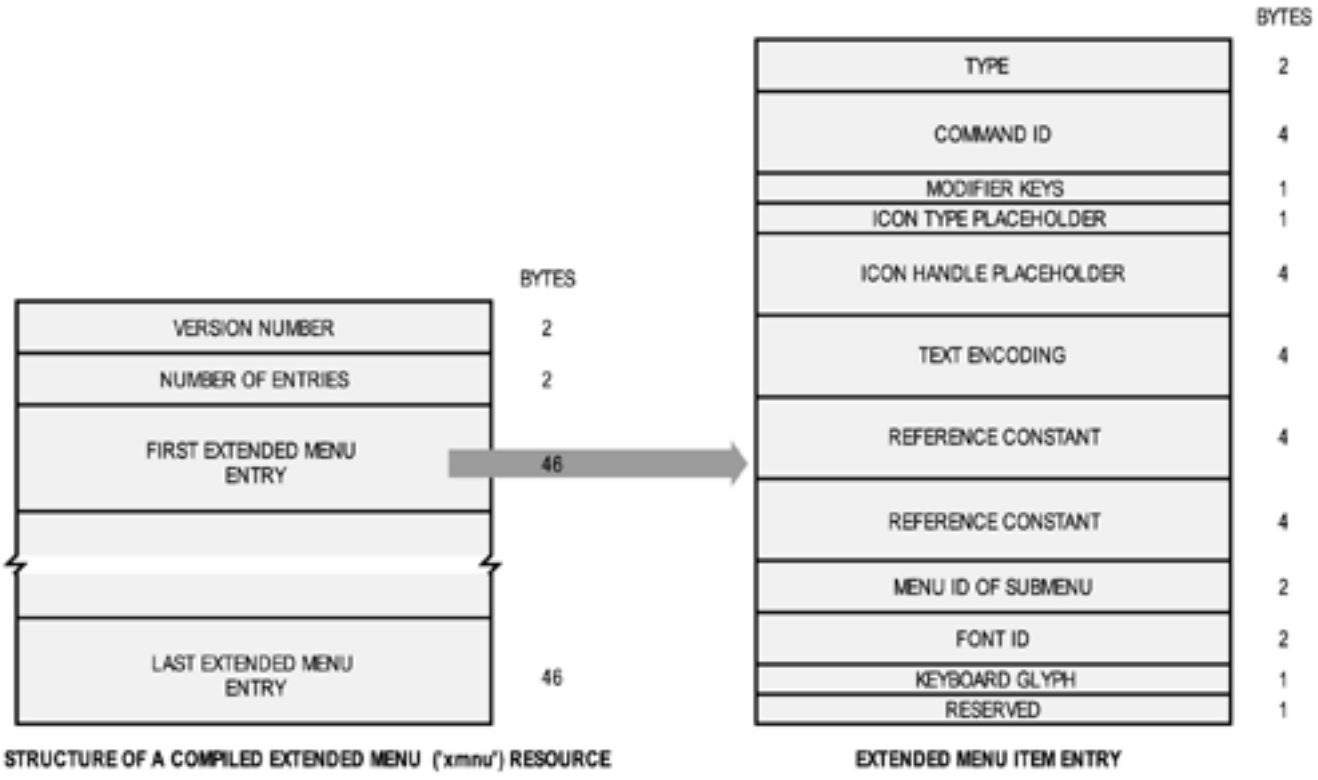
Note

As the user traverses menu items, if an item has a submenu, the `MenuSelect` function looks in the submenu portion of the menu list for the submenu. It then searches for a menu with a defined menu ID that matches the menu ID specified by the hierarchical menu item. If it finds a match, it attaches the submenu to the menu item.

Structure of a Compiled 'xmnu' Resource

The `'xmnu'` resource provides for the additional features, for example, support for extended modifier keys, command IDs, and so on. Fig 7 shows the structure of a compiled `'xmnu'` resource and an individual menu item entry in that resource.

Figure 7. STRUCTURE OF A COMPILED EXTENDED MENU ('xmnu') RESOURCE AND AN EXTENDED MENU ITEM ENTRY



The following describes the fields of a compiled `'xmnu'` resource:

Field	*Description*
VERSION NUMBER	Version of the resource.
NUMBER OF ENTRIES	Number of entries (extended menu item structures) in the resource.

FIRST EXTENDED MENU ENTRY	A number of extended menu item structures (see below).
…	
LAST EXTENDED MENU ENTRY	

Each entry in a `'xmnu'` resource corresponds to a menu item. The following describes the main fields of an extended menu item entry.

Field	***Description***
TYPE	Specifies whether there is extended information for the item. 1 indicates that there is extended information for the item, causing the rest of the entry to be read in. 0 indicates that there is no information for the item, causing the Menu Manager to skip the rest of the entry.
COMMAND ID	A unique value used to identify the menu item (as opposed to referring to the item using the menu ID and item number). This value may be ascertained via a call to `GetMenuItemCommandID`. A command ID may be assigned to a menu item programmatically via a call to `SetMenuItemCommandID`
MODIFIER KEYS	Specifies the modifier keys used in a keyboard equivalent to select a menu item. The current modifier keys may be ascertained via a call to `GetMenuItemModifiers`. Modifier keys may be assigned to a menu item programmatically via a call to `SetMenuItemModifiers`.
ICON TYPE PLACEHOLDER	(Reserved. Set to 0.)
ICON HANDLE PLACEHOLDER	(Reserved. Set to 0.)

TEXT ENCODING	Specifies the text encoding for the menu item text. This field of the `'xmnu'` resource should be used instead of setting the keyboard equivalent field in the `'MENU'` resource to `0x1C` and the icon number field to the script code. If you want to use the system script, assign -1. If you want to use the current script, assign -2. The current text encoding may be ascertained via a call to `GetMenuItemTextEncoding`. Text encoding may be assigned to a menu item programmatically via a call to `SetMenuItemTextEncoding`.
REFERENCE CONSTANT	Any value an application wants to store. The current value may be ascertained via a call to `GetMenuItemRefCon`. Reference constants may be assigned to a menu item programmatically via a call to `SetMenuItemRefCon`.
REFERENCE CONSTANT	Any additional value an application wants to store. The getter and setter functions relating to this second reference constant are not available in Carbon. If you wish to associate data with a menu item you should use the functions which are available for that purpose (see "Associating Data With Menu Items").

MENU ID OF SUBMENU	A value between 1 and 235, identifying the submenu. The current submenu ID may be acertained via a call to `GetMenuItemHierarchicalID`. The menu ID of a submenu may be assigned to a menu item programmatically via a call to `SetMenuItemHierarchicalID`. This, in effect, attaches a submenu to the menu item.
FONT ID	The ID of the font family. If this value is 0, then the large system font ID (Mac OS 8/9) or system font (Mac OS X) is used. The current font ID may be acertained via a call to `GetMenuItemFontID`. The font ID of a menu item may be set programmatically via a call to `SetMenuItemFontID`.
KEYBOARD GLYPH	A symbol representing a menu item's modifier key. The current keyboard glyph may be ascertained via a call to `GetMenuItemGlyph`. If the value in this field is zero, the keyboard glyph uses the keyboard font. (A glyph is a visual representation of a character.) You can override the character code to be displayed with a substitute glyph by assigning a non-zero value to this field. The keyboard glyph of a menu item may be set programmatically via a call to `SetMenuItemKeyGlyph`.

The information in an `'xmnu'` resource is set for specified menu items; it is not necessary to create an extended menu entry for all menu items in a menu.

It is not necessary to provide `'xmnu'` resources if your application's menus do not require the additional features provided by this resource.

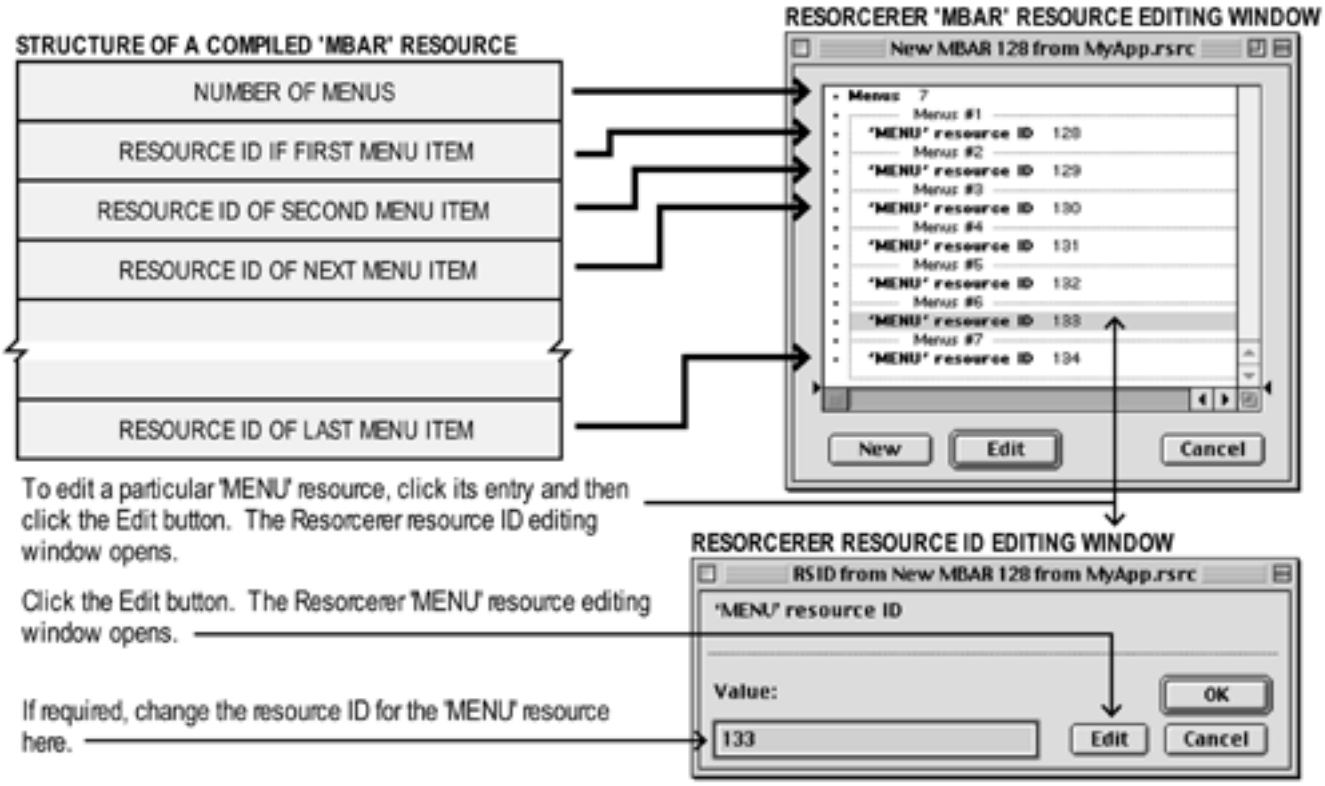
Creating 'MBAR', 'MENU', and 'xmnu' Resources Using Resorcerer

As previously stated, when creating resources using Resorcerer, it is advisable that you refer to a diagram and description of the structure of the resource and relate that to the various items in the Resorcerer editing windows. The following assumes that approach.

Creating 'MBAR' Resources

Fig 8 shows an 'MBAR' resource containing seven menus being created with Resorcerer. The first three entries would be, respectively, the Apple, File, and Edit menus.

Figure 8. CREATING AN 'MBAR' RESOURCE USING RESORCERER


STRUCTURE OF A COMPILED 'MBAR' RESOURCE
NUMBER OF MENUS
RESOURCE ID IF FIRST MENU ITEM
RESOURCE ID OF SECOND MENU ITEM
RESOURCE ID OF NEXT MENU ITEM
RESOURCE ID OF LAST MENU ITEM
RESORCERER 'MBAR' RESOURCE EDITING WINDOW
New MBAR 128 from MyApp.rsrc
Menus 7
Menus #1
'MENU' resource ID 128
Menus #2
'MENU' resource ID 129
Menus #3
'MENU' resource ID 130
Menus #4
'MENU' resource ID 131
Menus #5
'MENU' resource ID 132
Menus #6
'MENU' resource ID 133
Menus #7
'MENU' resource ID 134
New
Edit
Cancel
To edit a particular 'MENU' resource, click its entry and then click the Edit button. The Resorcerer resource ID editing window opens.
Click the Edit button. The Resorcerer 'MENU' resource editing window opens.
If required, change the resource ID for the 'MENU' resource here.
RESORCERER RESOURCE ID EDITING WINDOW
RSID from New MBAR 128 from MyApp.rsrc
'MENU' resource ID
Value:
133
OK
Edit
Cancel

Creating 'MENU' Resources

Fig 9 shows an imaginary View menu with the Full Screen menu item being edited. This menu item has been assigned a keyboard equivalent (more specifically, a Command-key equivalent); accordingly, the Key Equiv: radio button has been clicked and the character F has been entered as the Command-key equivalent. The menu item has also been assigned a marking character (a checkmark).

Figure 9. EDITING A 'MENU' RESOURCE USING RESORCERER

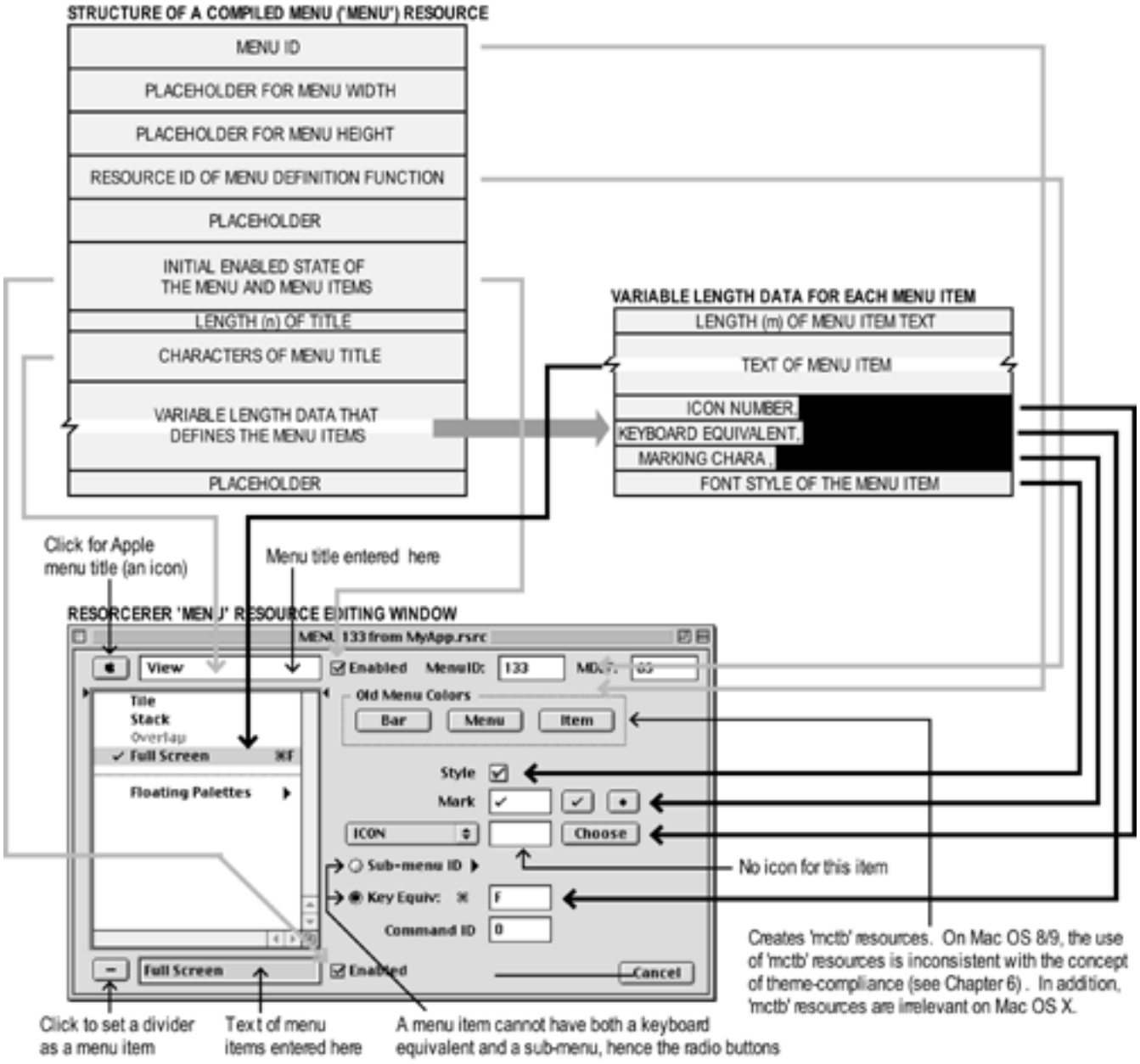
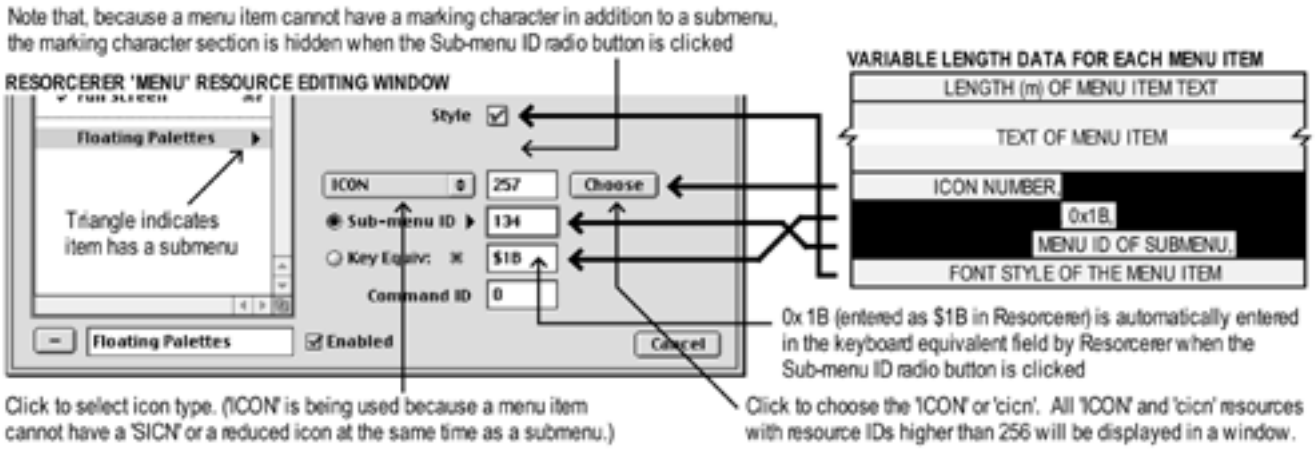


Fig 10 shows the same View menu with the Floating Palettes menu item being edited. This item has a submenu; accordingly, the Sub-menu ID radio button has been clicked and the resource ID of the submenu's `'MENU'` resource has been entered. The item also has an icon provided by a `'CICN'` or `'cicn'` resource with a resource ID of 257.

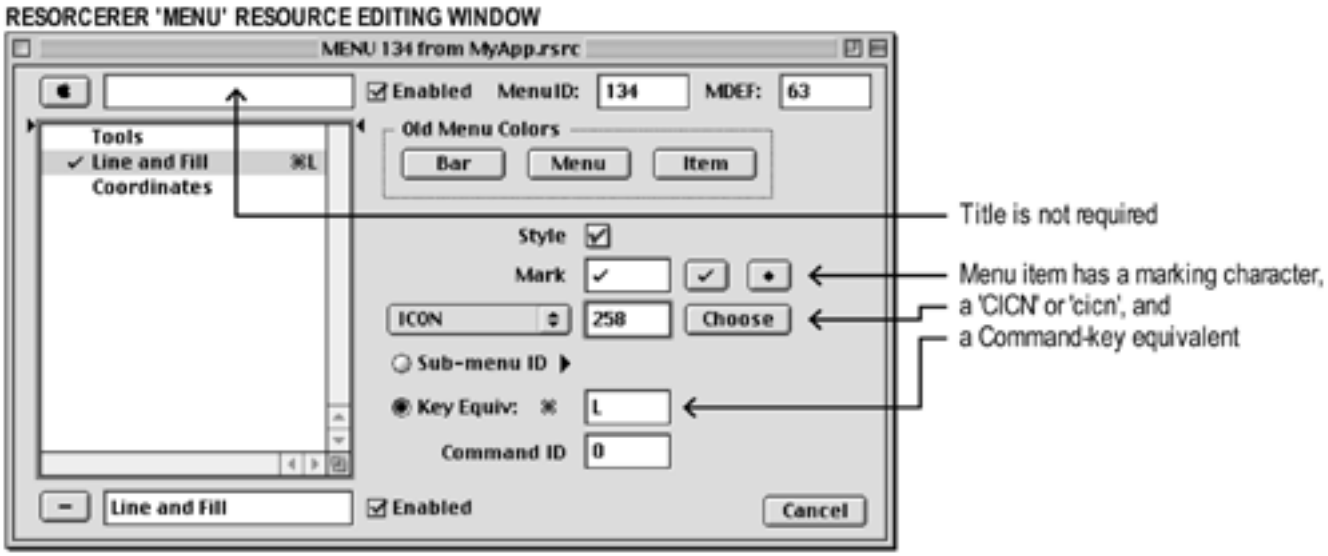
Figure 10. FURTHER EDITING OF A 'MENU' RESOURCE USING RESORCERER


Note that, because a menu item cannot have a marking character in addition to a submenu, the marking character section is hidden when the Sub-menu ID radio button is clicked
RESORCERER 'MENU' RESOURCE EDITING WINDOW
VARIABLE LENGTH DATA FOR EACH MENU ITEM
LENGTH (m) OF MENU ITEM TEXT
TEXT OF MENU ITEM
ICON NUMBER,
0x1B,
MENU ID OF SUBMENU,
FONT STYLE OF THE MENU ITEM
Floating Palettes
Triangle indicates item has a submenu
Style
ICON
257
Choose
Sub-menu ID
134
Key Equiv:
$1B
Command ID
0
Floating Palettes
Enabled
Cancel
0x 1B (entered as $1B in Resorcerer) is automatically entered in the keyboard equivalent field by Resorcerer when the Sub-menu ID radio button is clicked
Click to select icon type. (ICON is being used because a menu item cannot have a SICN or a reduced icon at the same time as a submenu.)
Click to choose the ICON or cicn. All ICON and cicn resources with resource IDs higher than 256 will be displayed in a window.

Creating 'MENU' Resources for Submenus

Fig 11 shows the Line and Fill submenu item in the submenu attached to the Floating Palettes menu item being edited. This item has a marking character (a checkmark), an icon provided by an `'ICON'` or `'cicn'` resource with resource ID 258, and a Command-key equivalent.

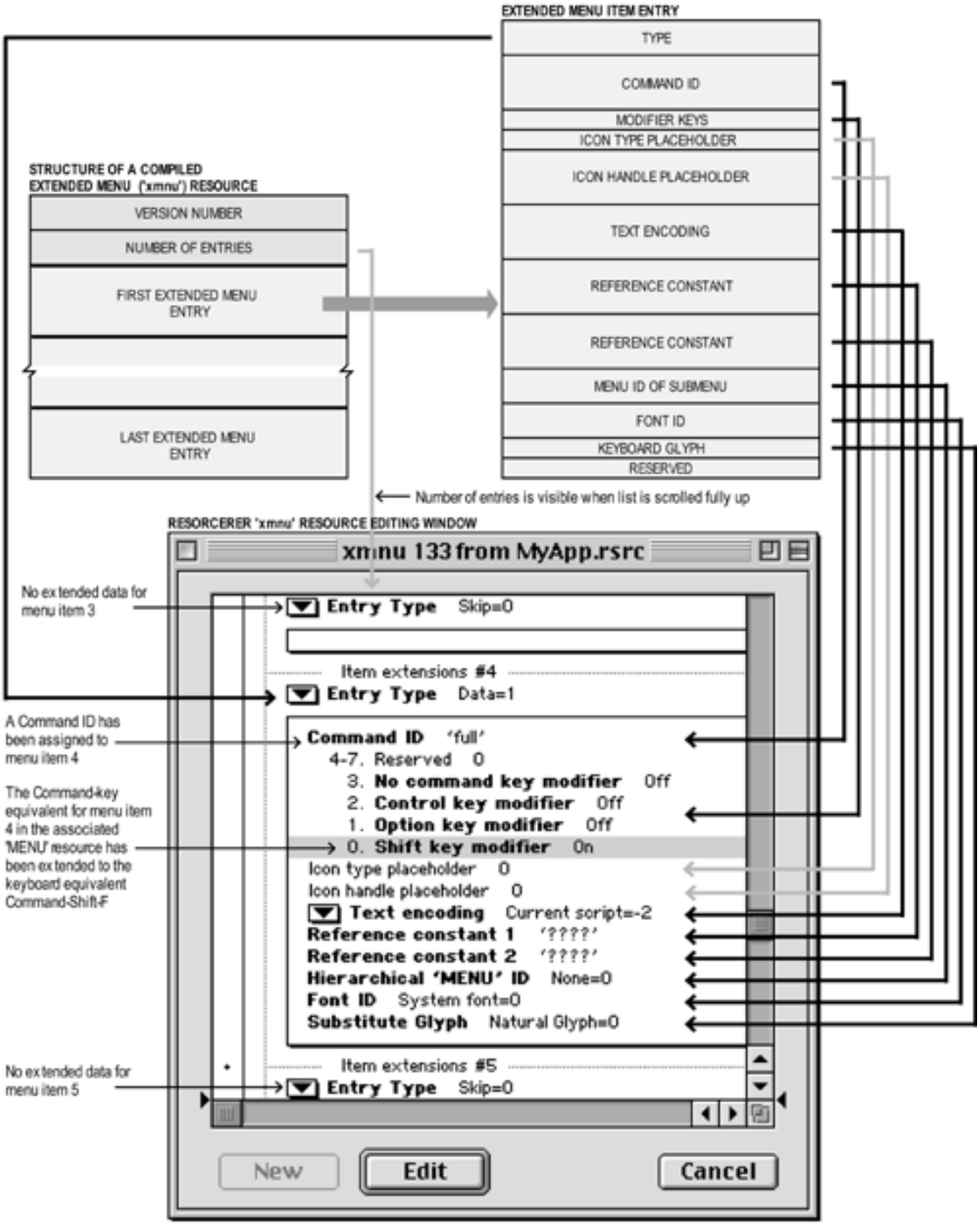
Figure 11. EDITING A 'MENU' RESOURCE FOR A SUBMENU USING RESORCERER



Creating 'xmnu' Resources

Fig 12 shows an `'xmnu'` resource being created using Resorcerer. This `'xmnu'` resource extends the `'MENU'` resource with resource ID 133 (the View menu, above). Menu item 4 has been assigned a command ID, and the Command-key equivalent assigned to this item in the `'MENU'` resource (Command-F) has been extended to the keyboard equivalent Command-Shift-F by specifying the Shift key as an extended modifier.

Figure 12. CREATING AN 'xmenu' RESOURCE USING RESORCERER


EXTENDED MENU ITEM ENTRY
TYPE
COMMAND ID
MODIFIER KEYS
ICON TYPE PLACEHOLDER
ICON HANDLE PLACEHOLDER
TEXT ENCODING
REFERENCE CONSTANT
REFERENCE CONSTANT
MENU ID OF SUBMENU
FONT ID
KEYBOARD GLYPH
RESERVED
STRUCTURE OF A COMPILED EXTENDED MENU ('xmnu') RESOURCE
VERSION NUMBER
NUMBER OF ENTRIES
FIRST EXTENDED MENU ENTRY
LAST EXTENDED MENU ENTRY
Number of entries is visible when list is scrolled fully up
RESORCERER 'xmnu' RESOURCE EDITING WINDOW
xmnu 133 from MyApp.rsrc
No extended data for menu item 3
Entry Type Skip=0
Item extensions #4
Entry Type Data=1
A Command ID has been assigned to menu item 4
Command ID 'full'
4-7. Reserved 0
3. No command key modifier Off
2. Control key modifier Off
1. Option key modifier Off
The Command-key equivalent for menu item 4 in the associated 'MENU' resource has been extended to the keyboard equivalent Command-Shift-F
0. Shift key modifier On
Icon type placeholder 0
Icon handle placeholder 0
Text encoding Current script=-2
Reference constant 1 '????'
Reference constant 2 '????'
Hierarchical 'MENU' ID None=0
Font ID System font=0
Substitute Glyph Natural Glyph=0
No extended data for menu item 5
Item extensions #5
Entry Type Skip=0
New
Edit
Cancel

Creating the Menu Bar and Pull-Down Menus

The function `GetNewMBar`, which itself calls `GetMenu`, should be used to read in the

'MBAR' resource and its associated 'MENU' resources. After reading in a 'MENU' resource, GetMenu looks for an 'xmnu' resource with the same resource ID and reads it in if found. GetNewMBar creates a menu object for each menu and inserts each menu into the menu list.

SetMenuBar should then be used to set the **current menu list** as the menu list created by your application. A call to DrawMenuBar completes the process by drawing the menu bar, displaying all the menu titles in the current menu list.

Deleting the Quit Command

As previously stated, your application must include a Quit command in the File menu when your application is run on Mac OS 8/9 but not when it is run on Mac OS X. Accordingly, you must conditionalize your code so as to ensure that the Quit command and its preceding divider are deleted when your application is run on Mac OS X. The methodology recommended by Apple is as follows:

```
SInt32  response;
MenuRef menuRef;

Gestalt(gestaltMenuMgrAttr,&response);
if(response & gestaltMenuMgrAquaLayoutMask)
{
  menuRef = GetMenuRef(mFile);
  if(menuRef != NULL)
  {
    DeleteMenuItem(menuRef,iQuit);
    DeleteMenuItem(menuRef,iQuit - 1);
  }
}
```

Creating a Hierarchical Menu

GetNewMBar does not read in the resource descriptions of submenus but simply records the menu ID of any submenu in the menu object. Submenu descriptions are read in with GetMenu and the submenu is inserted in the current menu list using InsertMenu, with the constant hierMenu passed as the second parameter to that call.

> **Note**
>
> As the user traverses menu items, if an item has a submenu, the MenuSelect function looks in the submenu portion of the menu list for the submenu. It then searches for a menu with a defined menu ID that matches the menu ID specified by the hierarchical menu item. If it finds a match, it attaches the submenu to the menu item.

Carbon Note

In Carbon, calling `GetMenu` twice on the same resource ID will create two independent, unique menus. (In Classic, the second call to `GetMenu` returns the same `MenuHandle` as the first call.) Thus, to prevent memory leaks in Carbon, `GetMenu` should not be called a second time on the same resource ID without an intervening call to `DisposeMenu`.

Adding Menus to the Menu List

A menu may be added to the current menu list using one of the following procedures:

- Read the relevant `'MENU'` resource in with `GetMenu`, add it to the current menu list with `InsertMenu`, and update the menu bar with `DrawMenuBar`.

- Use `NewMenu` to create a new empty menu, use `AppendMenu`, `InsertMenuItem`, `InsertResMenu`, or `AppendResMenu` to fill the menu with menu items, add the menu to the current menu list using `InsertMenu`, and update the menu bar using `DrawMenuBar`.

Note that `GetMenuRef` may be used to obtain a reference to the menu object of any menu in the current menu list.

Providing Help Balloons (Mac OS 8/9)

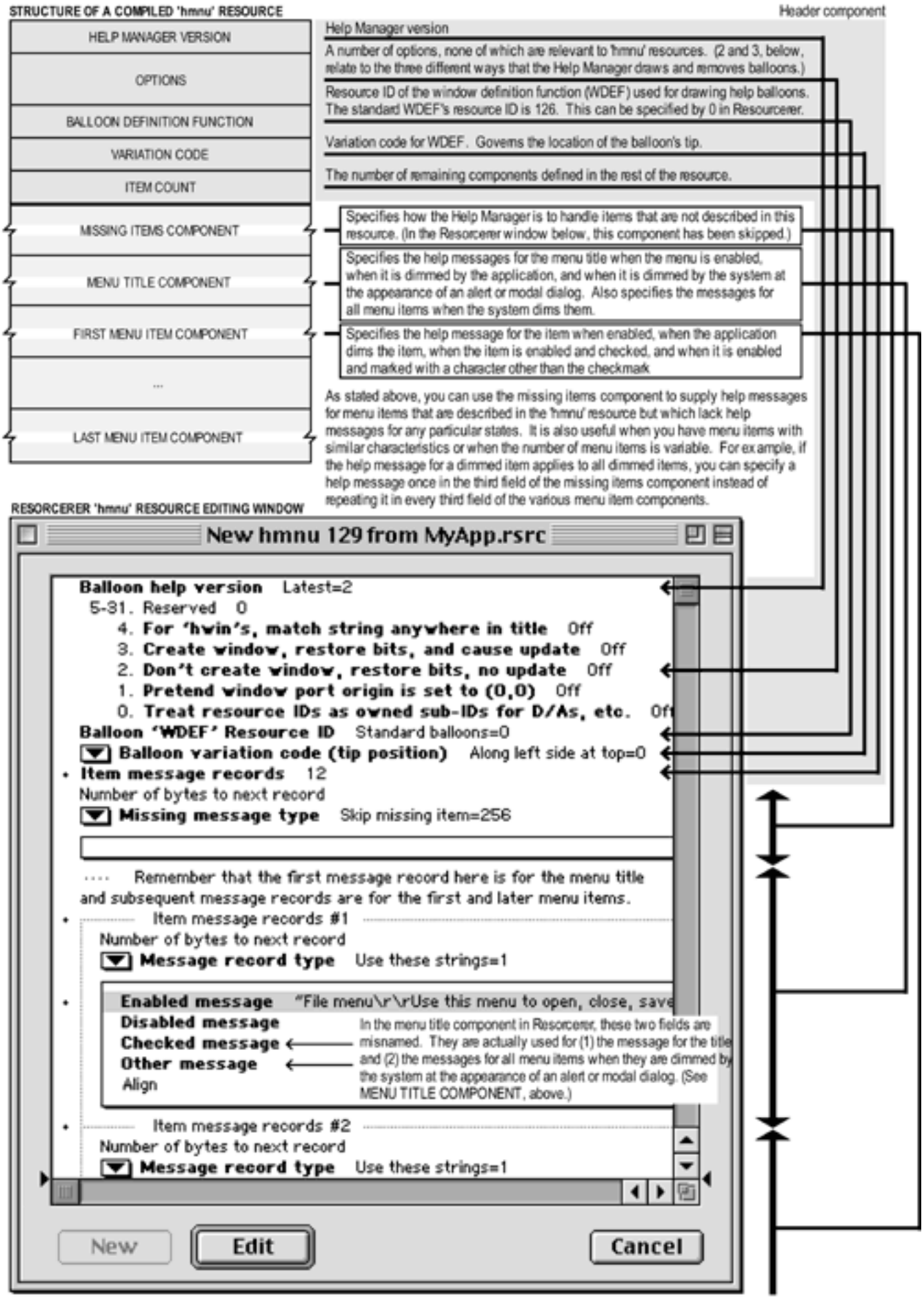
'hmmu' Resources

For Mac OS 8/9, you should define Help balloons for each of your application's menu items and each menu title. Help balloons for menus are defined in `'hmmu'` resources. The resource ID of an `'hmnu'` resource should be the same as the resource ID of the `'MENU'` resource to which it pertains.

Creating 'hmnu' Resources

Fig 13 shows an `'hmnu'` (help menu) resource being created using Resorcerer.

Figure 13. CREATING AN 'hmnu' RESOURCE USING RESORCERER


STRUCTURE OF A COMPILED 'hmnu' RESOURCE
HELP MANAGER VERSION
OPTIONS
BALLOON DEFINITION FUNCTION
VARIATION CODE
ITEM COUNT
MISSING ITEMS COMPONENT
MENU TITLE COMPONENT
FIRST MENU ITEM COMPONENT
...
LAST MENU ITEM COMPONENT
Header component
Help Manager version
A number of options, none of which are relevant to 'hmnu' resources. (2 and 3, below, relate to the three different ways that the Help Manager draws and removes balloons.)
Resource ID of the window definition function (WDEF) used for drawing help balloons. The standard WDEF's resource ID is 126. This can be specified by 0 in Resourcerer.
Variation code for WDEF. Governs the location of the balloon's tip.
The number of remaining components defined in the rest of the resource.
Specifies how the Help Manager is to handle items that are not described in this resource. (In the Resorcerer window below, this component has been skipped.)
Specifies the help messages for the menu title when the menu is enabled, when it is dimmed by the application, and when it is dimmed by the system at the appearance of an alert or modal dialog. Also specifies the messages for all menu items when the system dims them.
Specifies the help message for the item when enabled, when the application dims the item, when the item is enabled and checked, and when it is enabled and marked with a character other than the checkmark
As stated above, you can use the missing items component to supply help messages for menu items that are described in the 'hmnu' resource but which lack help messages for any particular states. It is also useful when you have menu items with similar characteristics or when the number of menu items is variable. For example, if the help message for a dimmed item applies to all dimmed items, you can specify a help message once in the third field of the missing items component instead of repeating it in every third field of the various menu item components.
RESORCERER 'hmnu' RESOURCE EDITING WINDOW
New hmnu 129 from MyApp.rsrc
Balloon help version Latest=2
5-31. Reserved 0
4. For 'hwin's, match string anywhere in title Off
3. Create window, restore bits, and cause update Off
2. Don't create window, restore bits, no update Off
1. Pretend window port origin is set to (0,0) Off
0. Treat resource IDs as owned sub-IDs for D/As, etc. Off
Balloon 'WDEF' Resource ID Standard balloons=0
Balloon variation code (tip position) Along left side at top=0
Item message records 12
Number of bytes to next record
Missing message type Skip missing item=256
.... Remember that the first message record here is for the menu title and subsequent message records are for the first and later menu items.
Item message records #1
Number of bytes to next record
Message record type Use these strings=1
Enabled message "File menu\r\rUse this menu to open, close, save
Disabled message
Checked message
Other message
Align
In the menu title component in Resorcerer, these two fields are misnamed. They are actually used for (1) the message for the title and (2) the messages for all menu items when they are dimmed by the system at the appearance of an alert or modal dialog. (See MENU TITLE COMPONENT, above.)
Item message records #2
Number of bytes to next record
Message record type Use these strings=1
New
Edit
Cancel

Specifying the Format of Help Messages.

The example at Fig 13 specifies the format of the help messages as (Pascal) text strings stored within the `'hmnu'` resource itself. Clicking on the pop-up button adjacent to Message record type opens a pop-up menu that facilitates the choice of other formats (and also provides an option that enables you to instruct the Help Manager to skip the item). The items in the pop-up menu and their meanings are as follows:

Pop-up Menu Item	***Meaning to Help Manager***
Use these strings	Use the strings specified within this component of this `'hmnu'` resource.
Use 'PICT' resources	Use the picture stored in the specified `'PICT'` resource.
Use 'STR#' resources	Use the specified text string stored in the specified `'STR#'` resource. (Storing the text strings in `'STR#'` resources or `'STR '` resources (see below) simplifies the task of providing foreign language versions of your application.)
Used styled text resources	Use the styled text stored in the specified `'TEXT'` and `'styl'` resources.
Use 'STR ' resources	Use the text string stored in the specified `'STR '` resource.
Use named resource type	Use the resource (`'STR '`, `'PICT'` or `'TEXT'`) whose name matches the name and state of the current menu item.
Skip missing item	No help message. Skip this item.
Compare item	Compare the specified comparison string against the current menu item in that position. If the specified string matches the name of the current menu item, display the help messages specified in the next four elements. (This is useful in the case of menu items that change names, for example Show Hidden Text and Hide Hidden Text.)

Text for Help Balloons

The text of your help balloons for menus should answer at least one of the following questions:

- ***What is this?*** For example, when the user moves the cursor over the title of the File menu in the title bar, the beginning of the balloon text should be "File menu".

- ***What does this do?*** For example, when the user moves the cursor over the Find item in a File menu, the balloon text should be "Finds and selects items with the characteristics you specify" or similar.

Changing Menu Item Appearance

Menu Manager functions may be used to change the appearance of items in a menu, for example, the font style, text or other characteristics.

Enabling and Disabling Menu Items

Specific menu items or entire menus may be disabled and enabled using `DisableMenuItem` and `EnableMenuItem`, which both take a reference to the menu object that identifies the desired menu and either the item number of the menu to be enabled/disabled or a value of 0 to indicate that the entire menu is to be enabled/disabled. Alternatively, if your application uses command IDs to identify menu items, you should use the functions `EnableMenuCommand` and `DisableMenuCommand` to enable and disable items.

When an entire menu is disabled or enabled, `DrawMenuBar` should be called to update the appearance of the menu bar. If you do not need to update the menu bar immediately, you can use `InvalMenuBar` instead of `DrawMenuBar`, causing the Event Manager to redraw the menu bar the next time it scans for update events. This will reduce the menu bar flicker that will occur if `DrawMenuBar` is called more than once in rapid succession.

If you disable an entire menu, the Menu Manager dims that menu's title at the next call to `DrawMenuBar` and dims all menu items when it displays the menu. If you enable an entire menu, the Menu Manager enables only the menu title and any items that you did not previously disable individually.

Enabling the Preferences… Item in the Application Menu

The Preferences… item in the Mac OS X Application menu is disabled by default. If your application needs to allow the user to invoke a Preferences dialog by choosing this item, it must explicitly enable the item. The following shows how to enable the Preferences… item:

```
EnableMenuCommand(NULL,kHICommandPreferences);
```

Other Appearance Changes

The following lists other functions related to changing the appearance of menu items.

Function	*Description*

`SetMenuItemText`
`SetMenuItemTextWithCFString`
`GetMenuItemText`
	Set and get the text.

`SetItemStyle` `GetItemStyle`	Set and get the font style.
`SetItemMark` `GetItemMark`	Set and get the marking character.
`SetItemIcon` `GetItemIcon`	Set and get the icon (`'ICON'` or `'cicn'`) using a resource ID.
`CheckMenuItem`	Places and removes a checkmark at the left of the item text.
`SetMenuItemFontID` `GetMenuItemFontID`	Set and get the font. `SetMenuItemFontID` allows you to set up a font menu with each item being drawn in the actual font.
`SetMenuItemIconHandle` `GetMenuItemIconHandle`	Set and get the icon (icon suite, `'ICON'` or `'cicn'`) using an icon handle. Provides, in conjuction with the `'xmnu'` resource, the support for icon suites introduced with Mac OS 8.
`SetMenuItemKeyGlyph` `GetMenuItemKeyGlyph`	`SetMenuItemKeyGlyph` substitutes a keyboard glyph for that normally displayed for a menu item's keyboard equivalent. `GetMenuItemKeyGlyph` gets the keyboard glyph for the keyboard equivalent.
`SetMenuFont` `GetMenuFont`	Set and get the font used in an individual menu.
`SetMenuExcludesMarkColumn` `GetMenuExcludesMarkColumn`	Set and get whether an individual menu contains space for marking characters.

Adding Items to a Menu

Adding Items Other Than the Names of Resources

`AppendMenu`, `InsertMenuItem`, `AppendMenuItemText`, `AppendMenuItemTextWithCFString`, `InsertMenuItemText` and `InsertMenuItemTextWithCFString` are used to add items other than the names of resources (such as font resources) to a previously created menu. They require:

- A reference to the menu object of the menu involved.
- A string describing the items to add.

Strings with Metacharacters

`AppendMenu` and `InsertMenuItem` allow you to specify the same characteristics for menu items as are available when defining a `'MENU'` resource. The string consists of the text of the menu item and any required characteristics. You can specify a hyphen as the menu item text to create a divider. You can also use various **metacharacters** in the text string to separate menu items and to specify the required characteristics. The following metacharacters may be used:

MetaCharacter	***Description***
`;` or Return	Separates menu items.
`^`	When followed by an icon number, defines the icon for the item.
`!`	When followed by a character, defines the mark for the item. If the keyboard equivalent field contains 0x1B, this value is interpreted as the menu ID of a submenu of this menu item.1
`<`	When followed by one or more of the characters `B`, `I`, `U`, `O`, and `S`, defines the character style of the item to, respectively, bold, italic, underline, outline or shadow.

/	When followed by a character, defines the Command-key equivalent for the item.[2] When followed by `0x1B`, specifies that this menu item has a submenu.[1] (Note: To specify that a menu item has a script code, reduced icon or small icon, use SetItemCmd to set the keyboard equivalent field to, respectively, `0x1C`, `0x1D` or `0x1E`.[3])
(	Defines the menu item as disabled.

As an example of the use of metacharacters, assume that the following two strings are stored in a string list (`'STR#'`) resource:

```
Pick a Color...
(^2!=Everything<B/E
```

The second string in this resource uses metacharacters to specify that the menu item is to be disabled, that it has an icon with a resource ID 258 (2+256), that it has the "=" character as a marking character, that the text style is bold, and that the item has a Command-key equivalent of Command-E.

> **Note**
>
> The Menu Manager adds 256 to the number you specify, and uses the result as the icon's resource ID.

Examples

The following code uses `AppendMenu` to append a menu item with no specific characteristics other than its text to the menu identified by the menu reference. The text for the menu item is "`Pick a Color...`" as stored in the preceding `'STR#'` resource.

```
MenuRef menuRef;
Str255  itemString;
...
menuRef = GetMenuRef(mLibrary);
GetIndString(itemString,300,1);
AppendMenu(menuRef,itemString);
```

To insert an item after a given menu item, use `InsertMenuItem`. The following code inserts the menu item "`Everything`" after the menu item with the item number specified in the `iRed` constant:

```
MenuRef menuRef;
Str255  ItemString;
...
menuRef = GetMenuRef(mColors);
GetIndString(itemString,300,2);
InsertMenuItem(menuRef,itemString,iRed);
```

The following code appends multiple items to the Edit menu using `AppendMenu`:

```
MenuRef menuRef;
...
menuRef = GetMenuRef(mEdit);
AppendMenu(menuRef,"\pUndo/Z;-;Cut/X;Copy/C;Paste/V");
```

`InsertMenuItem` differs from `AppendMenu` in the way it handles the given text string when that string contains multiple items, inserting them in reverse order. This code is equivalent to the last line of the preceding code:

```
InsertMenuItem(menuRef,"\pPaste/V;Copy/C;Cut/X-;-;Undo/Z",0);
```

The following code adds a divider to the Edit menu:

```
AppendMenu(menuRef,"\p(-");
```

Strings Without Metacharacters

`AppendMenuItemText`, `AppendMenuItemTextWithCFString`, `InsertMenuItemText` and `InsertMenuItemTextWithCFString` append and insert the specified string without evaluating the string for metacharacters. These functions may be used if you have a need to present non-alphanumeric characters in a menu item.

Adding Items Comprising Resource Names to a Menu

`AppendResMenu` or `InsertResMenu` may be used to add items that consist of resource names to a menu. For example, you can use `AppendResMenu` to add the names of all resident fonts as menu items in your application's Font menu.

[2] *When* `'xmnu'` *resources are used, use* `SetMenuItemModifiers` *to set the extended modifier keys (Shift, Option, Control).*

[1] *Applicable only to menus without* `'xmnu'` *resources. When* `'xmnu'` *resources are used, use* `SetMenuItemHierarchicalID` *to attach a submenu to a menu item.*

[3] *Applicable only to menus without* `'xmnu'` *resources. When* `'xmnu'` *resources are used, do not use* `SetItemCmd` *to specify a script code. Use* `SetMenuItemTextEncoding`.

Associating Data with Menu Items

The following functions may be used by your application to associate data with menu items:

Function	*Description*
`SetMenuItemProperty`	Associates data with a menu item.
`GetMenuItemPropertySize`	Obtains the size of a piece of data that has been previously associated with a menu item.
`GetMenuItemProperty`	Obtains a piece of data that has been previously associated with a menu item.
`RemoveMenuItemProperty`	Removes a piece of data that has been previously associated with a menu item.

Note that these functions require a command ID to be passed in their second parameter. Accordingly, they may only be used when your application uses command IDs to refer to the relevant menu items.

Handling Menu Choices

Determining the Menu ID and Menu Item-MenuSelect and MenuEvent

The first action your application should take when the user presses the mouse button while the cursor is in the menu bar is menu adjustment (that is, enabling or disabling menu items and adding or removing marks as required). Your application should then call `MenuSelect`. `MenuSelect` tracks the mouse, displays menus, highlights menu titles, displays and highlights enabled menu items, handles all user activity until the user releases the mouse button, and returns a long integer as its function result. The long integer contains the menu ID in the high word and the item number in the low word.

If some of your menu items have keyboard equivalents, your application should detect such key-down events. If an examination of the `modifiers` field of the event structure reveals that the Command key was down, your application should first adjust its menus and then call `MenuEvent`. `MenuEvent` scans the current menu list for a menu item that has a matching keyboard equivalent. Like `MenuSelect`, `MenuEvent` returns a long integer containing the menu ID and the item number.

If the user did not actually choose a menu command with the mouse, or if the user pressed a keyboard combination that did not map to a keyboard equivalent, `MenuSelect` and `MenuEvent` return 0 in the high word, the value in the low word being undefined.

Further Handling-Command IDs Not Used

The long integer returned by `MenuSelect` and `MenuEvent` should be passed as a parameter to a function that switches according to the menu ID in the high word and passes the low word to other functions that respond appropriately to that menu command.

Further Handling-Command IDs Used

Mac OS 8 introduced an alternative method of identifying, for the purposes of further handling, the menu item chosen by the user. This method assumes that you have previously assigned a unique value to your individual menu items via the command ID field of the `'xmnu'` resource (or, programmatically, via calls to `SetMenuItemCommandID`).

Using this method, the menu ID and item number should be extracted from the long integer returned by `MenuSelect` and `MenuEvent` in the usual way. The menu ID should then be used in a call to `GetMenuRef` to get the reference to the menu's menu object. This reference and the menu item should then be used in a call to `GetMenuItemCommandID`, which returns the unique value that you previously assigned to the item (that is, the item's command ID). Your application should then switch according to that command ID, calling the other functions that respond appropriately to that menu command.

Unhighlighting the Menu Title

Recall that one of the actions of `MenuSelect` and `MenuEvent` is to highlight the menu title. Ordinarily, your application should not unhighlight the menu title (using `HiliteMenu`) until it performs the action associated with the menu command chosen by the user. However, if, in response to a menu command, your application displays a modal dialog containing an edit text item, you should unhighlight the menu title immediately so that the user can access the Edit menu.

Adjusting Menus

Menu adjustment should be on the basis of the type of window that is currently the frontmost window, for example, a text window, a modeless dialog, and so on. Accordingly, the menu adjustment function should first determine which window is the front window. The following are examples of menu adjustment functions:

```
void  doAdjustMenus(void)
{
  WindowRef windowRef;
  SInt16    windowType;

  windowRef = FrontWindow();
```

```
  windowType = doGetWindowType(windowRef);

  switch windowType
  {
    case kMyDocWindow:
      doAdjustFileMenuForDocWindow();
      doAdjustEditMenuForDocWindow();
      // Adjust others.
      break;

    case kMyModelessDialogWindow:
      doAdjustMenusForModelessDialogs();
      break;

    case kNil:
      doAdjustMenusNoWindows();
      break;
  } ;

  DrawMenuBar;
}

void doAdjustFileMenuForDocWindow(void)
{
  MenuRef menuRef;

  menuRef = GetMenuRef(mFile);

  EnableMenuItem (menuRef,iNew);
  EnableMenuItem (menuRef,iOpen);
  DisableMenuItem(menuRef,iClose);
  DisableMenuItem(menuRef,iSave);
  DisableMenuItem(menuRef,iSaveAs);
  DisableMenuItem(menuRef,iPageSetup);
  DisableMenuItem(menuRef,iPrint);
  EnableMenuItem (menuRef,iQuit);
}
```

Handling Mac OS 8/9 Apple Menu Choices

When the user chooses an item in the Mac OS 8/9 Apple menu, `MenuSelect` returns the menu ID of your application's Apple menu in the high word and the item number in the low word.

If your application provides an About command as the first menu item in the Apple menu, and the user chooses this item, you should display the About dialog/alert. Other choices from the Mac OS 8/9 Apple menu are handled automatically by the system.

Handling Mac OS X Application Menu Choices

About Command

When the user chooses the About command in the Mac OS X Application menu, `MenuSelect` returns the menu ID of your application's Application menu in the high word and 1 in the low word. Thus the code that handles the user's choice of the Apple menu's About item on Mac OS 8/9 will also handle the user's choice of the Application menu's About item on Mac OS X.

Quit Command

When the user chooses the Quit command from the Mac OS X Application menu, a high-level event (more specifically, an Apple event) known as a Quit Application event is sent to your application. Your application must support the Quit Application event in order to respond to the user choosing the Quit command (see Chapter 10).

Preferences… Command

An Apple Event, known as the Show Preferences event, is sent to your application when the user chooses the Preferences… command from the Mac OS X Application menu. Your application must support the Show Preferences event in order to respond to the user choosing the Preferences…command. (See the demonstration program in Chapter 19.)

Handling a Size Menu

Preamble

On Mac OS 8/9, font sizes in Size menus should be outlined to indicate which sizes are directly provided by the current font. For bitmapped fonts, you should outline only those sizes that exist in the Fonts folder. For TrueType fonts, all sizes supported by that font should be outlined. The current font size should be indicated with a checkmark. If the current selection contains more than one font size, a dash should be placed next to each font size in the selection.

Size menus should, in addition to displaying available font sizes, provide an Other command to enable the user to specify a size not currently listed in the menu. When the user chooses the Other command, the current font size should be displayed in a dialog which allows the user to enter the desired font size. If the user chooses a size not already in the menu, a checkmark should be added to the Other menu item and the chosen size should be added in parenthesis to the text of the Other command.

Handling the Menu Choice

The following is an example function that handles a user's choice of an item in a Size menu:

```
void  doHandleSizeCommand(SInt16 menuItem)
{
  SInt16  numItems;
  Boolean addItem;
  SInt32  sizeChosen;

  numItems = CountMenuItems(GetMenuHandle(mSize));
  if(menuItem == numItems)          // If user chose Other, display dialog. If the
  {                                  // user-specified size is not in the menu, add a
    doDisplayOtherBox(sizeChosen);  // checkmark to the Other command and add the new
  }                                  // font size to the text of the Other command.
  else                              // Return sizeChosen.
  {                                  // User chose a size.
    doRemoveMarksFromSizeMenu();    // Remove marks from item/s showing previous size.
    CheckMenuItem(GetMenuHandle(mSize),menuItem,true);  // Add mark to chosen item.
    sizeChosen = doItemToSize(menuItem);     // Convert item number to font size.
  }
  doResizeSelection(sizeChosen);             // Update document state or user selection.
}
```

Hiding and Showing the Menu Bar

`HideMenuBar` and `ShowMenuBar` may be used to make the menu bar invisible and unselectable and visible and selectable. On Mac OS X, these functions also hide and show the Dock.

Accessing Menus from Alerts and Dialogs

When alerts and dialogs are displayed, the Dialog Manager and the Menu Manager interact to provide varying degrees of access to menus in your menu bar. In some circumstances, you can rely on the system software to disable the appropriate menus and menu items. In other circumstances, you application must contribute to, or control, the matter of menu access.

The subject of menu access when alerts, movable alerts, modal dialogs, moveable modal dialogs, and modeless dialogs are displayed is somewhat involved, and is addressed in detail in Chapter 8.

Main Menu Manager Constants, Data Types, and Functions

Constants

For markChar Parameter of SetItemMark Calls

```
noMark       = 0
commandMark  = 17
checkMark    = 18
diamondMark  = 19
appleMark    = 20
```

For beforeID Parameter of InsertMenu to Insert a Submenu into the Submenu Portion of the Menu List

```
hierMenu     = -1
```

For options Parameter of CreateStandardFontMenu Calls

```
KHierarchicalFontMenuOption  = 0x00000001
```

Modifier Key Masks for GetMenuItemModifiers and SetMenuItemModifiers Calls

```
kMenuCommandModifiers  = 0         If no bit is set, only the Command key is used.
kMenuShiftModifier     = (1 << 0)  If this bit (bit 0) is set, the Shift key is used.
kMenuOptionModifier    = (1 << 1)  If this bit (bit 1) is set, the Option key is used.
kMenuControlModifier   = (1 << 2)  If this bit (bit 2) is set, the Control key is used.
kMenuNoCommandModifier = (1 << 3)  If this bit (bit 3) is set, the Command key is not used.
```

Menu Icon Handle Constants for GetMenuItemIconHandle and SetMenuItemIconHandle Calls

```
kMenuNoIcon            = 0  No icon.
kMenuIconType          = 1  'ICON' handle.
kMenuShrinkIconType    = 2  32-by-32 'ICON' handle shrunk (at display time) to 16-by-16.
kMenuSmallIconType     = 3  'SICN' handle.
kMenuColorIconType     = 4  'cicn' handle.
kMenuIconSuiteType     = 5  Icon suite handle.
```

Menu Attributes

```
kMenuAttrExcludesMarkColumn = (1 << 0)
kMenuAttrAutoDisable        = (1 << 2)
```

Data Types

```
typedef struct OpaqueMenuHandle*  MenuHandle;
typedef MenuHandle  MenuRef;
typedef SInt16      MenuID;
typedef UInt16      MenuItemIndex;
typedef UInt32      MenuCommand;
typedef Handle      MenuBarHandle;
typedef UInt32      MenuAttributes;
```

Functions

Creating and Disposing of Menus

```
MenuRef   NewMenu(MenuID menuID, ConstStr255Param menuTitle);
MenuRef   GetMenu(short resourceID);
```

```
void      DisposeMenu(MenuRef theMenu);
```

Creating a Help Menu (Mac OS X)

```
OSStatus  HMGetHelpMenu(MenuRef *outHelpMenu,MenuItemIndex *outFirstCustomItemIndex);
```

Adding Menus to and Removing Menus from the Current Menu List

```
void      InsertMenu(MenuRef theMenu,MenuID beforeID);
void      DeleteMenu(MenuID menuID);
void      ClearMenuBar(void);
```

Getting a MenuBar Description from an 'MBAR' resource

```
MenuBarHandle  GetNewMBar(short menuBarID);
```

Getting, Setting and Disposing of the Menu Bar

```
MenuBarHandle  GetMenuBar(void);
void           SetMenuBar(MenuBarHandle mbar);
OSStatus       DisposeMenuBar(MenuBarHandle mbar);
short          GetMBarHeight(void);
```

Drawing the Menu Bar

```
void      DrawMenuBar(void);
void      InvalMenuBar(void);
```

Controlling Menu Bar Visibility

```
void      HideMenuBar(void);
void      ShowMenuBar(void);
Boolean   IsMenuBarVisible(void);
```

Modifying the Menu Width

```
Boolean   GetMenuExcludesMarkColumn(MenuRef menu);
OSStatus  SetMenuExcludesMarkColumn(MenuRef menu,Boolean excludesMark);
```

Responding to User Choice of a Menu Command

```
Uint32    MenuEvent(const EventRecord *inEvent);
long      MenuSelect(Point startPt);
long      MenuChoice(void);
void      HiliteMenu(MenuID menuID);
long      PopUpMenuSelect(MenuRef menu,short top,short left,short popUpItem);
```

Getting a Reference to a Menu object

```
MenuRef   GetMenuRef(MenuID menuID);
```

Adding and Deleting Menu Items

```
void      AppendMenu(MenuRef menu,ConstStr255Param data);
void      InsertMenuItem(MenuRef theMenu,ConstStr255Param itemString,short afterItem);
OSStatus  AppendMenuItemText(MenuRef menu,ConstStr255Param inString);
OSStatus  AppendMenuItemTextWithCFString(MenuRef menu,CFStringRef inString,
          MenuItemAttributes inAttributes,MenuCommand inCommandID,MenuItemIndex
*outNewItem);
OSStatus  InsertMenuItemText(MenuRef menu,ConstStr255Param inString,UInt16 afterItem);
OSStatus  InsertMenuItemTextWithCFString(MenuRef menu,CFStringRef inString,
```

```
          MenuItemIndex inAfterItem,MenuItemAttributes inAttributes,MenuCommand
inCommandID);
void      DeleteMenuItem(MenuRef theMenu,short item);
void      AppendResMenu(MenuRef theMenu,ResType theType);
void      InsertResMenu(MenuRef theMenu,ResType theType,short afterItem);
```

Enabling and Disabling Menus, Menu Items, and Menu Item Icons

```
void      EnableMenuItem(MenuRef theMenu,MenuItemIndex item);
void      DisableMenuItem(MenuRef theMenu,MenuItemIndex item);
Boolean   IsMenuItemEnabled(MenuRef menu,MenuItemIndex item);
void      DisableAllMenuItems(MenuRef theMenu);
void      EnableAllMenuItems(MenuRef theMenu);
Boolean   MenuHasEnabledItems(MenuRef theMenu);
void      EnableMenuCommand(MenuRef theMenu,MenuCommand commandID);
void      DisableMenuCommand(MenuRef theMenu,MenuCommand commandID);
Boolean   IsMenuCommandEnabled(MenuRef menu,MenuCommand commandID);
void      EnableMenuItemIcon(MenuRef theMenu,MenuItemIndex item);
void      DisableMenuItemIcon(MenuRef theMenu,MenuItemIndex item);
Boolean   IsMenuItemIconEnabled(MenuRef menu,MenuItemIndex item);
```

Getting and Setting Menu Item Command IDs

```
OSErr     GetMenuItemCommandID(MenuRef inMenu,SInt16 inItem,UInt32 *outCommandID);
OSErr     SetMenuItemCommandID(MenuRef inMenu,SInt16 inItem,UInt32 inCommandID);
```

Menu Object Accessor Functions

```
MenuID    GetMenuID(MenuRef menu);
void      SetMenuID(MenuRef menu,MenuID menuID);
SInt16    GetMenuWidth(MenuRef menu);
void      SetMenuWidth(MenuRef menu,SInt16 width);
SInt16    GetMenuHeight(MenuRef menu);
void      SetMenuHeight(MenuRef menu,SInt16 height);
StringPtr GetMenuTitle(MenuRef menu,Str255 title);
OSStatus  SetMenuTitle(MenuRef menu,ConstStr255Param title);
OSStatus  SetMenuTitleWithCFString(MenuRef menu,CFStringRef inString);
OSStatus  GetMenuDefinition(MenuRef menu,MenuDefSpecPtr outDefSpec);
OSStatus  SetMenuDefinition (MenuRef menu,const MenuDefSpec *defSpec);
```

Manipulating and Accessing Menu Item Characteristics

```
void      GetMenuItemText(MenuRef menu,short item,Str255 itemString);
void      SetMenuItemText(MenuRef theMenu, short item, ConstStr255Param itemString);
OSStatus  SetMenuItemTextWithCFString(MenuRef menu,MenuItemIndex item,CFStringRef inString);
void      GetItemStyle(MenuRef theMenu,short item,Style* chStyle);
void      SetItemStyle(MenuRef theMenu,short item, StyleParameter chStyle);
void      GetItemMark(MenuRef theMenu,short item,CharParameter *markChar);
void      SetItemMark(MenuRef theMenu,short item,CharParameter markChar);
void      CheckMenuItem(MenuRef theMenu,short item,Boolean checked);
OSStatus  GetMenuFont(MenuRef menu,SInt16 *outFontID,UInt16 *outFontSize);
OSStatus  SetMenuFont(MenuRef menu,SInt16 inFontID,UInt16 inFontSize);
void      GetItemIcon(MenuRef theMenu,short item,short *iconIndex);
void      SetItemIcon(MenuRef theMenu,short item,short iconIndex);
void      GetItemCmd(MenuRef theMenu,short item,short *cmdChar);
void      SetItemCmd(MenuRef theMenu,short item,short cmdChar);
OSErr     GetMenuItemFontID(MenuRef inMenu,SInt16 inItem,SInt16* outFontID);
OSErr     SetMenuItemFontID(MenuRef inMenu,SInt16 inItem,SInt16 inFontID);
OSErr     GetMenuItemHierarchicalID(MenuRef inMenu,SInt16 inItem,SInt16 *outHierID);
OSErr     SetMenuItemHierarchicalID(MenuRef inMenu,SInt16 inItem,SInt16 inHierID);
OSErr     GetMenuItemIconHandle(MenuRef inMenu,SInt16 inItem,MenuIconType outIconType,
          Handle* outIconHandle);
OSErr     SetMenuItemIconHandle(MenuRef inMenu,SInt16 inItem,MenuIconType inIconType,
          Handle inIconHandle);
```

```
OSErr     GetMenuItemKeyGlyph(MenuRef inMenu,SInt16 inItem,SInt16 *outGlyph);
OSErr     SetMenuItemKeyGlyph(MenuRef inMenu,SInt16 inItem,SInt16 inGlyph);
OSErr     GetMenuItemModifiers(MenuRef inMenu,SInt16 inItem,SInt16* outModifiers);
OSErr     SetMenuItemModifiers(MenuRef inMenu,SInt16 inItem,SInt16 inModifiers);
OSErr     GetMenuItemRefCon(MenuRef inMenu,SInt16 inItem,SInt32* outRefCon);
OSErr     SetMenuItemRefCon(MenuRef inMenu,SInt16 inItem,SInt32 inRefCon);
OSErr     GetMenuItemTextEncoding(MenuRef inMenu,SInt16 inItem,TextEncoding* outScriptID);
OSErr     SetMenuItemTextEncoding(MenuRef inMenu,SInt16 inItem,TextEncoding inScriptID);
```

Getting a Menu Reference and Menu ID from a Command ID

```
OSStatus  GetIndMenuItemWithCommandID(MenuRef menu,MenuCommand commandID,UInt32 itemIndex,
          MenuRef *outMenu,MenuItemIndex *outIndex);
```

Associating Data with Menu Items

```
OSStatus  SetMenuCommandProperty(MenuRef menu,MenuCommand commandID,OSType propertyCreator,
          OSType propertyTag,ByteCount propertySize,const void *propertyData);
OSStatus  GetMenuCommandPropertySize(MenuRef menu,MenuCommand commandID,OSType
propertyCreator,
          OSType propertyTag,ByteCount *size);
OSStatus  GetMenuCommandProperty(MenuRef menu,MenuCommand commandID,OSType propertyCreator,
          OSType propertyTag,ByteCount bufferSize,ByteCount *actualSize,void
*propertyBuffer);
OSStatus  RemoveMenuCommandProperty(MenuRef menu,MenuCommand commandID,OSType
propertyCreator,
          OSType propertyTag);
```

Counting Items in a Menu

```
short     CountMenuItems(MenuRef theMenu);
ItemCount CountMenuItemsWithCommandID(MenuRef menu,MenuCommand commandID);
```

Building and Updating Font Menus and Obtaining Font Family and Style

```
OSStatus  CreateStandardFontMenu(MenuRef menu,MenuItemIndex afterItem,MenuID
firstHierMenuID,
          OptionBits options,ItemCount *outHierMenuCount);
OSStatus  UpdateStandardFontMenu(MenuRef menu,ItemCount *outHierMenuCount);
OSStatus  GetFontFamilyFromMenuSelection(MenuRef menu,MenuItemIndex item,
          FMFontFamily *outFontFamily,FMFontStyle *outStyle);
```

Recalculating Menu Dimensions

```
void      CalcMenuSize(MenuRef theMenu);
```

Highlighting the Menu Bar

```
void      FlashMenuBar(MenuID menuID);
```

Demonstration Program Menus1 Listing

```
// ****************************************************************************************
// Menus1.c                                                          CLASSIC EVENT MODEL
// ****************************************************************************************
//
// This program:
//
// •  Opens a window.
//
// •  Creates these pull-down menus: Apple, File, Edit, Font, Size, Special, and Window.
//
// •  Displays text in the window indicating the menu selection made by the user.
//
// The Apple menu includes an "About..." menu item for the program.
//
// The second menu item in the Special menu contains a submenu.
//
// The Font and Window menus are created programmatically using the functions
// CreateStandardFontMenu and CreateStandardWindowMenu.
//
// The implementation of the Size menu is nominal only. The current size is indicated with a
// checkmark; however, the number of sizes shown is not font-dependent and there is no "Other"
// item.
//
// Because the primary purpose of the program is to demonstrate menu creation and handling, no
// code is included to update and activate/deactivate the window or to respond to events which
// are not relevant to the demonstration.
//
// The program is terminated by selecting Quit from the File menu, by pressing the keyboard
// equivalent for that item (Command-Q), or by clicking in the window's go-away box/close
// button.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  A 'WIND' resource (purgeable) (initially not visible).
//
// •  An 'MBAR' resource (preload, non-purgeable).
//
// •  'MENU' resources for the Apple, File, Edit, Font, Size, and Special drop-down menus
//    and the submenu (all preload, all non-purgeable).
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// ****************************************************************************************

// ..........................................................................................................................................................................................
includes

#include <Carbon.h>

// ..........................................................................................................................................................................................
defines

#define rMenubar          128
#define mAppleApplication 128
#define  iAbout           1
#define mFile             129
#define  iQuit            12
#define mEdit             130
#define  iUndo            1
#define  iCut             3
#define  iCopy            4
#define  iPaste           5
#define  iClear           6
#define mFont             131
#define mSize             132
#define  iTen             1
#define  iTwelve          2
#define  iEighteen        3
#define  iTwentyFour      4
#define mSpecial          133
#define  iFirstItem       1
#define  iSecondItem      2
#define mWindow           134
#define mSubmenu          135
#define mFirstFontSubMenu 136
#define  iFirstSubItem    1
#define  iSecondSubItem   2
#define rWindowResource   128

// ..................................................................................................................................................................... global variables
```

```
Boolean        gDone;
ItemCount      gFontMenuHierMenuCount;
MenuItemIndex  gCurrentFontSizeItem = 2;
MenuItemIndex  gCurrentFontMenuItem;
MenuItemIndex  gCurrentFontSubMenuItem;
MenuRef        gCurrentFontSubMenuRef = NULL;

// ..................................................................................................................................................................... function prototypes

void  main                  (void);
void  doPreliminaries       (void);
OSErr quitAppEventHandler   (AppleEvent *,AppleEvent *,SInt32);
void  doGetMenus            (void);
void  doEvents              (EventRecord *);
void  doMouseDown           (EventRecord *);
void  doAdjustMenus         (void);
void  doMenuChoice          (SInt32);
void  doAppleApplicationMenu (MenuItemIndex);
void  doFileMenu            (MenuItemIndex);
void  doEditMenu            (MenuItemIndex);
void  doFontMenu            (MenuID,MenuItemIndex);
void  doSizeMenu            (MenuItemIndex);
void  doSpecialMenu         (MenuItemIndex);
void  doSubMenus            (MenuItemIndex);
void  drawItemString        (Str255);

// ********************************************************************************** main

void  main(void)
{
  EventRecord eventStructure;
  WindowRef   windowRef;

  // ..................................................................................................................................................... do preliminaries

  doPreliminaries();

  // ..................................................................................................................................................... open a window
  if(!(windowRef = GetNewCWindow(rWindowResource,NULL,(WindowRef) -1)))
  {
    SysBeep(10);
    ExitToShell();
  }

  SetPortWindowPort(windowRef);

  // ..................................................................................................... set up menu bar and menus, then show window

  doGetMenus();
  ShowWindow(windowRef);

  // ..................................................................................................................................................... event loop

  gDone = false;

  while(!gDone)
  {
    if(WaitNextEvent(everyEvent,&eventStructure,180,NULL))
      doEvents(&eventStructure);
  }
}

// ************************************************************************** doPreliminaries

void  doPreliminaries(void)
{
  OSErr osError;

  MoreMasterPointers(640);
  InitCursor();
  FlushEvents(everyEvent,0);

  osError = AEInstallEventHandler(kCoreEventClass,kAEQuitApplication,
                          NewAEEventHandlerUPP((AEEventHandlerProcPtr) quitAppEventHandler),
                          0L,false);
  if(osError != noErr)
    ExitToShell();
}

// ************************************************************************** doQuitAppEvent

OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  OSErr    osError;
  DescType returnedType;
  Size     actualSize;

  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,NULL,0,
```

```
                            &actualSize);

  if(osError == errAEDescNotFound)
  {
    gDone = true;
    osError = noErr;
  }
  else if(osError == noErr)
    osError = errAEParamMissed;

  return osError;
}

// ********************************************************************************* doGetMenus

void  doGetMenus(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  OSStatus      osError;
  Str255        smallSystemFontName, itemString;
  SInt16        numberOfItems,a;

  // ..................................................................................................................................................................... get and set, and menu bar

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    ExitToShell();
  SetMenuBar(menubarHdl);

  // ................................ if running on Mac OS X, delete Quit item and preceding divider from File menu

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
      DisableMenuItem(menuRef,0);
    }
  }

  // ..................................................................................... create hirearchical Font menu, checkmark small system font

  menuRef = GetMenuRef(mFont);
  if(menuRef != NULL)
  {
    osError = CreateStandardFontMenu(menuRef,0,mFirstFontSubMenu,kHierarchicalFontMenuOption,
                                     &gFontMenuHierMenuCount);
    if(osError != noErr)
      ExitToShell();
  }
  else
    ExitToShell();

   GetFontName(kThemeSmallSystemFont,smallSystemFontName);
  numberOfItems = CountMenuItems(menuRef);
  for(a=1;a<numberOfItems + 1;a++)
  {
    GetMenuItemText(menuRef,a,itemString);
    if(EqualString(itemString,smallSystemFontName,false,false))
    {
      CheckMenuItem(menuRef,a,true);
      gCurrentFontMenuItem = a;
      break;
    }
  }

  // ..................................................................................................................................... create Window menu and insert into menu list

  CreateStandardWindowMenu(0,&menuRef);
  SetMenuID(menuRef,mWindow);
  InsertMenu(menuRef,0);

  // ..................................................................................................................................... get submenus and insert into window list

  menuRef = GetMenu(mSubmenu);
  if(menuRef != NULL)
    InsertMenu(menuRef,hierMenu);
  else
    ExitToShell();

  // ..................................................................... set initial font size and checkmark associated item in Size menu

  doSizeMenu(gCurrentFontSizeItem);
```

```
  // ..................................................................................................................................................................... draw menu bar

  DrawMenuBar();
 }

// ********************************************************************************* doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
  switch(eventStrucPtr->what)
  {
    case kHighLevelEvent:
      AEProcessAppleEvent(eventStrucPtr);
      break;

    case mouseDown:
      doMouseDown(eventStrucPtr);
      break;
    case keyDown:
      if((eventStrucPtr->modifiers & cmdKey) != 0)
      {
        doAdjustMenus();
        doMenuChoice(MenuEvent(eventStrucPtr));
      }
      break;

    case updateEvt:
      BeginUpdate((WindowRef) eventStrucPtr->message);
      EndUpdate((WindowRef) eventStrucPtr->message);
      break;
  }
}

// ********************************************************************************* doMouseDown

void  doMouseDown(EventRecord *eventStrucPtr)
{
  WindowRef      windowRef;
  WindowPartCode partCode;
  SInt32         menuChoice;

  partCode = FindWindow(eventStrucPtr->where,&windowRef);

  switch(partCode)
  {
    case inMenuBar:
      doAdjustMenus();
      menuChoice = MenuSelect(eventStrucPtr->where);
      doMenuChoice(menuChoice);
      break;

    case inContent:
      if(windowRef != FrontWindow())
        SelectWindow(windowRef);
      break;

    case inDrag:
      DragWindow(windowRef,eventStrucPtr->where,NULL);
      break;

    case inGoAway:
      if(TrackGoAway(windowRef,eventStrucPtr->where))
        gDone = true;
      break;
  }
}

// ********************************************************************************* doAdjustMenus

void  doAdjustMenus(void)
{
  // Adjust menus here.
}

// ********************************************************************************* doMenuChoice

void  doMenuChoice(SInt32 menuChoice)
{
  MenuID        menuID;
  MenuItemIndex menuItem;

  menuID    = HiWord(menuChoice);
  menuItem  = LoWord(menuChoice);

  if(menuID == 0)
    return;

  if(menuID == mFont || ((menuID >= mFirstFontSubMenu) &&
                         (menuID <= mFirstFontSubMenu + gFontMenuHierMenuCount)))
```

```
    doFontMenu(menuID,menuItem);
  else
  {
    switch(menuID)
    {
      case mAppleApplication:
        doAppleApplicationMenu(menuItem);
        break;

      case mFile:
        doFileMenu(menuItem);
        break;

      case mEdit:
        doEditMenu(menuItem);
        break;

      case mSize:
        doSizeMenu(menuItem);
        break;

      case mSpecial:
        doSpecialMenu(menuItem);
        break;

      case mSubmenu:
        doSubMenus(menuItem);
        break;
    }
  }

  HiliteMenu(0);
}
// ********************************************************************* doAppleApplicationMenu

void  doAppleApplicationMenu(MenuItemIndex menuItem)
{
  if(menuItem == iAbout)
    drawItemString("\pAbout Menus1");
}

// ************************************************************************************ doFileMenu

void  doFileMenu(MenuItemIndex menuItem)
{
  if(menuItem  == iQuit)
    gDone = true;
}

// ************************************************************************************ doEditMenu

void  doEditMenu(MenuItemIndex menuItem)
{
  switch(menuItem)
  {
    case iUndo:
      drawItemString("\pUndo");
      break;

    case iCut:
      drawItemString("\pCut");
      break;

    case iCopy:
      drawItemString("\pCopy");
      break;

    case iPaste:
      drawItemString("\pPaste");
      break;

    case iClear:
      drawItemString("\pClear");
      break;
  }
}

// ************************************************************************************ doFontMenu

void  doFontMenu(MenuID menuID,MenuItemIndex menuItem)
{
  MenuRef      menuRef, fontMenuRef;
  OSStatus     osError;
  FMFontFamily currentFontFamilyReference;
  FMFontStyle  currentFontStyle;
  Str255       fontName, styleName, itemName;
  SInt16       a, numberOfFontMenuItems;
  menuRef = GetMenuRef(menuID);
```

```
  osError = GetFontFamilyFromMenuSelection(menuRef,menuItem,&currentFontFamilyReference,
                                           &currentFontStyle);
  if(osError == noErr)
  {
    TextFont(currentFontFamilyReference);
    TextFace(currentFontStyle);

    GetFontName(currentFontFamilyReference,fontName);
    drawItemString(fontName);

    if(menuID == mFont)
    {
      CheckMenuItem(menuRef,gCurrentFontMenuItem,false);
      gCurrentFontMenuItem = menuItem;
      CheckMenuItem(menuRef,gCurrentFontMenuItem,true);

      if(gCurrentFontSubMenuRef != NULL)
        CheckMenuItem(gCurrentFontSubMenuRef,gCurrentFontSubMenuItem,false);
    }
    else
    {
      if(gCurrentFontSubMenuRef != NULL)
        CheckMenuItem(gCurrentFontSubMenuRef,gCurrentFontSubMenuItem,false);
      gCurrentFontSubMenuRef = menuRef;
      gCurrentFontSubMenuItem = menuItem;
      CheckMenuItem(gCurrentFontSubMenuRef,gCurrentFontSubMenuItem,true);

      fontMenuRef = GetMenuRef(mFont);
      CheckMenuItem(fontMenuRef,gCurrentFontMenuItem,false);

      numberOfFontMenuItems = CountMenuItems(fontMenuRef);

      for(a=1;a<=numberOfFontMenuItems;a++)
      {
        GetMenuItemText(fontMenuRef,a,itemName);
        if(EqualString(fontName,itemName,true,true))
        {
          gCurrentFontMenuItem = a;
          break;
        }
      }

      SetItemMark(fontMenuRef,gCurrentFontMenuItem,'-');

      GetMenuItemText(menuRef,menuItem,styleName);
      DrawString("\p ");
      DrawString(styleName);
    }
  }
  else
    ExitToShell();
}
// ********************************************************************************** doSizeMenu

void  doSizeMenu(MenuItemIndex menuItem)
{
  MenuRef sizeMenuRef;

  switch(menuItem)
  {
    case iTen:
      TextSize(10);
      break;

    case iTwelve:
      TextSize(12);
      break;

    case iEighteen:
      TextSize(18);
      break;

    case iTwentyFour:
      TextSize(24);
      break;
  }

  sizeMenuRef = GetMenuRef(mSize);

  CheckMenuItem(sizeMenuRef,gCurrentFontSizeItem,false);
  CheckMenuItem(sizeMenuRef,menuItem,true);

  gCurrentFontSizeItem = menuItem;

  drawItemString("\pSize change");
}

// ********************************************************************************** doSpecialMenu
```

```
void  doSpecialMenu(MenuItemIndex menuItem)
{
  if(menuItem == iFirstItem)
    drawItemString("\pFirst Item");
}

// ********************************************************************************* doSubMenus

void  doSubMenus(MenuItemIndex menuItem)
{
  switch(menuItem)
  {
    case iFirstSubItem:
      drawItemString("\pSubitem 1");
      break;
    case iSecondSubItem:
      drawItemString("\pSubitem 2");
      break;
  }
}

// ********************************************************************************* drawItemString

void  drawItemString(Str255 eventString)
{
  RgnHandle tempRegion;
  WindowRef windowRef;
  Rect      scrollBox;

  windowRef = FrontWindow();
  tempRegion = NewRgn();

  GetWindowPortBounds(windowRef,&scrollBox);

  ScrollRect(&scrollBox,0,-24,tempRegion);
  DisposeRgn(tempRegion);

  MoveTo(8,286);
  DrawString(eventString);
}

// ********************************************************************************************
```

Demonstration Program Menus1 Comments

When this program is run, the user should choose items from all menus, including the Apple menu. Selections should be made using the mouse and, where appropriate, the Command key equivalents. The user should also note the effects on the menu bar of clicking outside, then inside, the program's window, that is, of sending the program to the background and returning it to the foreground.

defines

Constants are established for the pull-down and submenu menu IDs and associated resource IDs, menu item numbers and subitem numbers.

The Menu Manager function `CreateStandardFontMenu` will be used to create a hierarchical Font menu and `mFirstFontSubMenu` establishes the ID of the first Font menu submenu to be created.

The last line establishes a constant for the resource ID of the `'WIND'` resource.

Global Variables

`gFontMenuHierMenuCount` will be assigned a value representing the number of submenus created by the Menu Manager function `CreateStandardFontMenu`.

`GCurrentFontSizeItem` will be assigned the menu item number of the chosen font size.

`gCurrentFontMenuItem` and `gCurrentFontSubMenuItem` will be used in the Font menu handling function to specify which menu and submenu items are to have a checkmark added or cleared. `gCurrentFontSubMenuRef` will be assigned a reference to the menu object for the currently chosen Font menu submenu.

main

The `main()` function creates a window and makes its graphics port the current port, calls `doGetMenus` to set up the menus, shows the window and enters the main event loop.

doPreliminaries

The large number of master pointers created by `MoreMasterPointers` in this program

allows for the likely creation of a large number of submenus by the Menu Manager function `CreateStandardFontMenu`.

When the program is run on Mac OS X, the Quit item will be in the Application menu. The system informs the program of the user's choice of this item via a high-level event known as an Apple event, more specifically, an Apple event known as the Quit Application event. The call to `AEInstallEventHandler` installs `quitAppEventHandler` as the handler for this high-level event. (Apple events and Apple event handlers are explained in Chapter 10.)

quitAppEventHandler

`quitAppEventHandler` is the handler for the Quit Application event installed in `doPreliminaries`. Basically, it sets the global variable `gDone` to true, which causes the program to terminate from the main event loop.

doGetMenus

`doGetMenus` sets up the menu bar and the various menus.

At the first block, `GetNewMBar` reads in the `'MENU'` resources for each menu specified in the `'MBAR'` resource and creates a menu object for each of those menus. (Note that the error handling here and in other areas of this program is somewhat rudimentary: the program simply terminates.) The call to `SetMenuBar` makes the newly created menu list the current list.

The call to Gestalt determines whether the application is running on Mac OS 8/9 or Mac OS X. If the application is running on Mac OS X, `GetMenuRef` is called to get a reference to the menu object for the File menu and `DeleteMenuItem` is called to delete the Quit item and its preceding divider from the menu.

The third block utilizes the relatively new Menu Manager function `CreateStandardFontMenu` to create a hierarchical font menu. A reference to the Font menu object is passed in the first parameter. The third parameter specifies the menu ID for the first submenu to be created. The fourth parameter specifies that the Font menu be created as a hierarchical menu. The fifth parameter receives a value representing the number of submenus created. `CreateStandardFontMenu` itself inserts these submenus into the submenu portion of the menu list.

The fourth block checkmarks the Font menu item containing the name of the small system font. `GetFontName` gets the name of the small system font and `CountMenuItems` counts the number of items in the Font menu. The for loop then walks the items in the Font menu looking for a match. When it finds a match, `CheckMenuItem` is called to checkmark the item, the global variable which keeps track of the currently selected font is

assigned the number of that item, and the for loop is exited.

The fifth block creates the Window menu using the Window Manager function `CreateStandardWindowMenu`. The accessor function `SetMenuID` sets the menu's ID and `InsertMenu` inserts the menu into the menu list. (Setting the menu ID is for illustrative purposes only because the ID is not used in this demonstration. Since the system handles the standard Window menu automatically, an ID is ordinarily only required for menu adjustment purposes when the menu has been customised.)

The sixth block inserts a further single submenu (to be attached to the second item in the Special menu) into the submenu portion of the menu list. `GetNewMBar` does not read in the resource descriptions of submenus, so the first step is to read in the `'MENU'` resource with `GetMenu`. `InsertMenu` inserts a menu reference for this menu into the menu list at the location specified in the second parameter to this call. Using the constant `hierMenu` (`-1`) as the second parameter causes the menu to be installed in the submenu portion of the menu list.

The last line causes a checkmark to be set against the Size menu item corresponding to the initialised value of the global variable `gCurrentFontSizeItem`.

`DrawMenuBar` draws the menu bar.

Note that, in Carbon, the contents of the Apple Menu Items folder are automatically added to the Apple menu.

doEvents

`doEvents` switches according to the type of low-level or Operating System event received. Further processing is called for in the case of mouse-down or Command key equivalents, these being central to the matter of menu handling.

At the `keyDown` case, a check is made of the modifiers field of the event structure to establish whether the Command key was also pressed at the time. If so, menu enabling/disabling is attended to before the call to `MenuEvent` establishes whether the character code is associated with a currently enabled menu or submenu item in the menu list. If a match is found, `MenuEvent` returns a long integer containing the menu ID in the high word and the item number in the low word, otherwise it returns 0 in the high word. This long integer is then passed to the function `doMenuChoice`.

doMouseDown

`doMouseDown` first establishes the window and window part in which the mouse-down event occurred, and switches accordingly. This demonstration program is specifically concerned with mouse-downs in the menu bar and the content region of the window.

If the event occurred in this program's menu bar, menu enabling/disabling is attended to before the call to `MenuSelect`. `MenuSelect` tracks the user's actions until the mouse button is released, at which time it returns a long integer. If the user actually chose a menu item, this long integer contains the menu ID in the high word and the item number in the low word, otherwise it contains 0 in the high word. This long integer is passed to the function `doMenuChoice`.

If the mouse-down event occurred in the content region of the window, and if the window to which the mouse-down refers is not the front window, `SelectWindow` is called to effect basic window activation/deactivation.

doAdjustMenus

`doAdjustMenus` is called when a mouse-down occurs in the menu bar and when examination of a key-down event reveals that a menu item's keyboard equivalent has been pressed. No action is required in this simple program.

Later demonstration programs contain examples of menu adjustment functions.

doMenuChoice

`doMenuChoice` takes the long integer returned by the `MenuSelect` and `MenuEvent` calls, extracts the high word (the menu ID) and the low word (the menu item number) and switches according to the menu ID.

At the first two lines, the menu ID and the menu item number are extracted from the long integer. The next two lines will cause an immediate return if the high word equals 0, (meaning that either the mouse button was released when the pointer was outside the menu box or `MenuEvent` found no menu list match for the key pressed in conjunction with the Command key).

If the menu ID represents either the Font menu or one of the Font menu's submenus, the menu handling function `doFontMenu` is called. Otherwise, the function switches on the menu ID so that the appropriate individual menu handling function is called. Note the handling of the submenu attached to the second item in the Special menu (case `mSubMenu`).

The Window menu is handled automatically by the system.

`MenuEvent` and `MenuSelect` leave the menu title highlighted if an item was actually chosen. Accordingly, the last line unhighlights the menu title when the action associated with the user's drop-down menu choice is complete.

doAppleApplicationMenu

`doAppleApplicationMenu` takes the short integer representing the menu item. If this value represents the first item in the Mac OS 8/9 Apple menu or Mac OS X Application menu (the inserted "About…" item), text representing this item is drawn in the scrolling display.

If other items in the Mac OS 8/9 Apple menu are chosen, the system automatically opens the chosen object and passes control that object.

doFileMenu

`doFileMenu` handles choices from the File menu when the program is run on Mac OS 8/9. In this demonstration, only the Quit item is enabled, all other items having been disabled in the File menu's `'MENU'` resource. When this item is chosen, the global variable `gDone` is set to true, causing termination of the program.

doEditMenu

`doEditMenu` switches according to the menu item number, drawing text representing the chosen item in the window.

doFontMenu

`doFontMenu` first gets a reference to the Font menu object. This, together with the menu item number, is passed in a call to the function `GetFontFamilyFromMenuSelection`. This function returns a reference to the font family and a value representing the font style. (A font family reference represents a collection of fonts with the same design characteristics, e.g., Arial, Arial Bold, Arial Italic, and Arial Bold Italic. Font style values are, for example, 0 = plain, 1 = bold, 2 = italic, 3 = bold italic).

The font family reference and the font style value are passed in calls to `TextFont` and `TextFace`, which will cause subsequent text drawing to be conducted in the specified font and style. The call to `GetFontName` gets the font's name from the font family reference and the function `drawItemString` draws that name in the window.

The remaining code is mainly concerned with checkmarking the newly chosen Font menu item and submenu item, and unchecking the previously chosen items.

If the menu ID represents the Font menu (meaning that a menu item without an attached submenu was chosen), the previously chosen item is unchecked, a global variable stores the item number of the newly-chosen item preparatory to the next call to `doFontmenu`, and the newly chosen item is checked. If a submenu item has previously been chosen, and thus

checked, it is unchecked.

If, on the other hand, the `menuID` represents one of the Font menu's submenus:

- If a submenu item has previously been chosen, that item is unchecked. A reference to the submenu object is assigned to a global variable, the menu item number is stored in another global variable preparatory to the next call to `doFontmenu`, and the newly chosen submenu item is checked.
- The next two lines uncheck the previously checked Font menu item.
- The for loop walks the Font menu looking for a match between item names and the font name previously extracted from the font family reference. When a match is found, the loop exits, the loop variable containing the item number where the match was found. This is stored in a global variable preparatory to the next call to `doFontMenu`, and is also passed in the call to `CheckMenuItem` to check that item.
- The last block gets the style name from the menu object and draws that next to the font name in the window.

doSizeMenu

`doSizeMenu` switches according to the menu item chosen in the Size menu, sets the text size for all text drawing to that size, unchecks the current size item, and checks the newly chosen item. `gCurrentSize` is then set to the chosen menu item number before the function returns.

doSpecialMenu

`doSpecialMenu` handles a choice of the first item in the Special menu. Since the second item is the title of a submenu, only the first item is attended to in this function.

doSubMenus

`doSubMenus` switches according to the chosen item in the submenu attached to the second menu item in the Special menu.

drawItemString

The function `drawItemString` is incidental to the demonstration, being called by the menu selection handling functions to draw text in the application's window to reflect the user's menu choices.

Demonstration Program Menus2 Listing

```
// ****************************************************************************************
// Menus2.c                                                          CLASSIC EVENT MODEL
// ****************************************************************************************
//
// This program is based on Menus1. The basic differences between this program and Menus1 are
// as follows:
//
// •  'xmnu' resources are used to extend the 'MENU' resources for some menus.
//
// •  Extended modifier keys (Shift, Option, and Control) are used to extend the Command-key
//    equivalents for two menu items in the Style menus.
//
// •  There are two Style menus (Style ('xmnu') and Style (Programmatic). The two Style menus
//    are intended to demonstrate assigning extended modifier keys to a menu item (1) via an
//    'xmnu' resource and (2)  programmatically.
//
// •  Command IDs are assigned to all menu items except those in the system-managed menus and
//    the Font menu, and the associated menu handling code branches according to the command
//    ID of the chosen menu item (as opposed to menu ID and menu item).
//
// •  The Font menu is non-hierarchical. It is also WYSIWYG, meaning that each item is drawn
//    in that font.
//
// •  The delete-to-the-left, delete-to-the-right, page-up, and page-down keys are assigned as
//    Command-key equivalents in the Size menu, and the glyphs are adjusted where necessary.
//
// •  The submenu is attached to the second item in the Special menu programmatically rather
//    than via the 'MENU' resource.
//
// •  Color icons are included in the menu items in the submenu.
//
// •  Balloon help is provided, via 'hmnu' resources, for all menus.
//
// The extended modifier keys in the Style ('xmnu') menu are assigned via the 'xmnu' resource
// for that menu. The extended  modifier keys in the Style (Programmatic) menu are assigned
// programmatically .
//
// The command IDs for items in the File, Edit, and Style ('xmnu') menus are assigned via the
// 'xmnu' resources for those menus. The command IDs for the items in the Style
// (Programmatic), Size, and Special menus, and the submenu, are assigned programmatically.
//
// The color icon in the first submenu item is assigned via the 'MENU' resource. The color
// icon in the second item is assigned programmatically via a call to
// SetMenuItemIconHandle.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  A 'WIND' resource (purgeable) (initially not visible).
//
// •  An 'MBAR' resource (preload, non-purgeable).
//
// •  'MENU' resources for the drop-down menus and submenu (all preload, all non-purgeable).
//
// •  'xmnu' resources (preload, purgeable) for the drop-down menus (except the system-managed
//     menus and the Font menu) and the submenu.
//
// •  'hmnu' resources (purgeable) providing balloon help for menus and menu items.
//
// •  Two 'cicn' resources (purgeable) for the items in the submenu.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// ****************************************************************************************

// ........................................................................................................................................................................................
includes

#include <Carbon.h>

// ........................................................................................................................................................................................
defines

#define rMenubar         128
#define mAppleApplication 128
#define mFile            129
#define  iQuit           12
#define mFont            131
#define mStyleXmnu       132
#define mStyleProg       133
#define  iPlain          1
```

```
#define  iBold           3
#define  iItalic         4
#define  iOutline        6
#define  iUnderline      5
#define  iShadow         7
#define mSize            134
#define  iTen            1
#define  iTwelve         2
#define  iEighteen       3
#define  iTwentyFour     4
#define mSpecial         135
#define  iFirst          1
#define  iSecond         2
#define mSubmenu         136
#define  iBat            1
#define  iBowl           2
#define rWindowResource   128
#define rColorIcon       258

// ..................................................................................................................................................................... global variables

Boolean       gRunningOnX          = false;
Boolean       gDone;
MenuItemIndex gCurrentFontMenuItem = 0;
Style         gCurrentStyle        = 0;
MenuItemIndex gCurrentSizeMenuItem = 2;

// ..................................................................................................................................................................... function prototypes

void  main                (void);
void  doPreliminaries     (void);
OSErr  quitAppEventHandler (AppleEvent *,AppleEvent *,SInt32);
void  doGetMenus          (void);
void  doEvents            (EventRecord *);
void  doMouseDown         (EventRecord *);
void  doAdjustMenus       (void);
void  doMenuChoice        (SInt32);
void  doCommand           (MenuCommand);
void  doFontMenu          (MenuItemIndex);
void  doCheckStyleMenuItem (MenuID);
void  doCheckSizeMenuItem  (MenuItemIndex);
void  drawItemString      (Str255);

// ********************************************************************************* main

void  main(void)
{
  EventRecord eventStructure;
  WindowRef   windowRef;
  RGBColor    foreColor = {  0xFFFF,0xFFFF,0xFFFF } ;
  RGBColor    backColor = {  0x4444,0x4444,0x9999 } ;
  Rect        portRect;

  // ..................................................................................................................................................................... do preliminaries

  doPreliminaries();

  // ..................................................................................................................................................................... open a window

  if(!(windowRef = GetNewCWindow(rWindowResource,NULL,(WindowRef) -1)))
  {
    SysBeep(10);
    ExitToShell();
  }

  SetPortWindowPort(windowRef);
  TextSize(10);
  RGBBackColor(&backColor);
  RGBForeColor(&foreColor);

  // ..................................................................................................... set up menu bar and menus, then show window

  doGetMenus();
  ShowWindow(windowRef);
  GetWindowPortBounds(windowRef,&portRect);
  EraseRect(&portRect);

  // ..................................................................................................................................................................... event loop

  gDone = false;

  while(!gDone)
  {
    if(WaitNextEvent(everyEvent,&eventStructure,180,NULL))
      doEvents(&eventStructure);
  }
}

// ************************************************************************** doPreliminaries
```

```
void  doPreliminaries(void)
{
  OSErr osError;

  MoreMasterPointers(32);
  InitCursor();
  FlushEvents(everyEvent,0);

  osError = AEInstallEventHandler(kCoreEventClass,kAEQuitApplication,
                          NewAEEventHandlerUPP((AEEventHandlerProcPtr) quitAppEventHandler),
                          0L,false);
  if(osError != noErr)
    ExitToShell();
}

// ********************************************************************************* doQuitAppEvent

OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  OSErr    osError;
  DescType returnedType;
  Size     actualSize;

  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,NULL,0,
                              &actualSize);

  if(osError == errAEDescNotFound)
  {
    gDone = true;
    osError = noErr;
  }
  else if(osError == noErr)
    osError = errAEParamMissed;

  return osError;
}

// ********************************************************************************** doGetMenus

void  doGetMenus(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  OSStatus      osError;
  ItemCount     hierMenuCount;
  SInt16        a, numberOfItems, fontNumber;
  Str255        fontName, smallSystemFontName;
  CIconHandle   cicnHdl;

  // ..................................................................................................................................................................... get and set menu bar

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    ExitToShell();
  SetMenuBar(menubarHdl);
  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
      DisableMenuItem(menuRef,0);
    }

    gRunningOnX = true;
  }

  // ..................................................................................................................................................... set up Font menu and make WYSIWYG

  GetFontName(kThemeSmallSystemFont,smallSystemFontName);

  menuRef = GetMenuRef(mFont);
  if(menuRef != NULL)
  {
    osError = CreateStandardFontMenu(menuRef,0,0,kNilOptions,&hierMenuCount);
    if(osError == noErr)
    {
      numberOfItems = CountMenuItems(menuRef);
      for(a=1;a<=numberOfItems;a++)
      {
        GetMenuItemText(menuRef,a,fontName);
        GetFNum(fontName,&fontNumber);
        SetMenuItemFontID(menuRef,a,fontNumber);

        if(EqualString(fontName,smallSystemFontName,false,false))
```

```
      {
        CheckMenuItem(menuRef,a,true);
        gCurrentFontMenuItem = a;
      }
    }
  }
  else ExitToShell();
}
else
  ExitToShell();

// ............................................... programmatically set the extended modifiers in Style (Programmatic) menu

menuRef = GetMenuRef(mStyleProg);
SetMenuItemModifiers(menuRef,iOutline,kMenuShiftModifier + kMenuOptionModifier
                     + kMenuControlModifier);
SetMenuItemModifiers(menuRef,iShadow,kMenuShiftModifier + kMenuOptionModifier);

// ............ insert submenu into menu list and programmatically attach it to Special menu, item 2

menuRef = GetMenu(mSubmenu);
if(menuRef != NULL)
{
  InsertMenu(menuRef,hierMenu);
  menuRef = GetMenuRef(mSpecial);
  SetMenuItemHierarchicalID(menuRef,iSecond,mSubmenu);
}
else
  ExitToShell();

// .................. programmatically set command IDs for second Style, Size, Special menus and submenu

menuRef = GetMenuRef(mStyleProg);
SetMenuItemCommandID(menuRef,iPlain,     'plai');
SetMenuItemCommandID(menuRef,iBold,      'bold');
SetMenuItemCommandID(menuRef,iItalic,    'ital');
SetMenuItemCommandID(menuRef,iUnderline, 'unde');
SetMenuItemCommandID(menuRef,iOutline,   'outl');
SetMenuItemCommandID(menuRef,iShadow,    'shad');

menuRef = GetMenuRef(mSize);
SetMenuItemCommandID(menuRef,iTen,       'ten ');
SetMenuItemCommandID(menuRef,iTwelve,    'twel');
SetMenuItemCommandID(menuRef,iEighteen,  'eigh');
SetMenuItemCommandID(menuRef,iTwentyFour,'twen');

menuRef = GetMenuRef(mSpecial);
SetMenuItemCommandID(menuRef,iFirst,     'firs');

menuRef = GetMenuRef(mSubmenu);
SetMenuItemCommandID(menuRef,iBat,       'bat ');
SetMenuItemCommandID(menuRef,iBowl,      'bowl');

// ..................................................................... programmatically set the icon for the Bowl item in the submenu

cicnHdl = GetCIcon(rColorIcon);
SetMenuItemIconHandle(menuRef,iBowl,kMenuColorIconType,(Handle) cicnHdl);

// ..................... programmatically set Command-key equivalents to Size menu items and adjust glyphs

menuRef = GetMenuRef(mSize);
SetItemCmd(menuRef,iTen,0x08);
SetMenuItemKeyGlyph(menuRef,iTen,kMenuDeleteLeftGlyph);
SetItemCmd(menuRef,iTwelve,0x7f);
SetMenuItemKeyGlyph(menuRef,iTwelve,kMenuDeleteRightGlyph);
SetItemCmd(menuRef,iEighteen,0x0b);
SetMenuItemKeyGlyph(menuRef,iEighteen,kMenuPageUpGlyph);
SetItemCmd(menuRef,iTwentyFour,0x0c);
SetMenuItemKeyGlyph(menuRef,iTwentyFour,kMenuPageDownGlyph);

// ............................... programmatically exclude the mark column and set the font in the Special menu

menuRef = GetMenuRef(mSpecial);
SetMenuExcludesMarkColumn(menuRef,true);
GetFNum("\pGadget",&fontNumber);
if(fontNumber != 0)
  SetMenuFont(menuRef,fontNumber,12);

// ..................................................................................... if running on Mac OS X, create Help menu and insert one item

if(gRunningOnX)
{
  HMGetHelpMenu(&menuRef,NULL);
  InsertMenuItem(menuRef,"\pMenus Help",0);
  SetMenuItemCommandID(menuRef,1,'help');
}

// ..................................................................................................... set initial font, style, and size, and checkmark them
```

```
  doCheckStyleMenuItem(mStyleXmnu);
  doCheckStyleMenuItem(mStyleProg);
  doCheckSizeMenuItem(iTen);

  // ..................................................................................................................................................................................... draw menu bar

  DrawMenuBar();
}

// ********************************************************************************* doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
  switch(eventStrucPtr->what)
  {
    case kHighLevelEvent:
      AEProcessAppleEvent(eventStrucPtr);
      break;

    case mouseDown:
      doMouseDown(eventStrucPtr);
      break;

    case keyDown:
      if((eventStrucPtr->modifiers & cmdKey) != 0)
      {
        doAdjustMenus();
        doMenuChoice(MenuEvent(eventStrucPtr));
      }
      break;

    case updateEvt:
      BeginUpdate((WindowRef) eventStrucPtr->message);
      EndUpdate((WindowRef) eventStrucPtr->message);
      break;
  }
}
// ********************************************************************************* doMouseDown

void  doMouseDown(EventRecord *eventStrucPtr)
{
  WindowRef      windowRef;
  WindowPartCode partCode;
  SInt32         menuChoice;

  partCode = FindWindow(eventStrucPtr->where,&windowRef);

  switch(partCode)
  {
    case inMenuBar:
      doAdjustMenus();
      menuChoice = MenuSelect(eventStrucPtr->where);
      doMenuChoice(menuChoice);
      break;

    case inContent:
      if(windowRef != FrontWindow())
        SelectWindow(windowRef);
      break;

    case inDrag:
      DragWindow(windowRef,eventStrucPtr->where,NULL);
      break;

    case inGoAway:
      if(TrackGoAway(windowRef,eventStrucPtr->where))
        gDone = true;
      break;
  }
}

// ********************************************************************************* doAdjustMenus

void  doAdjustMenus(void)
{
  // Adjust menus here. Use EnableMenuCommand and DisableMenuCommand to enable/disable those
  // menu items with command IDs.
}

// ********************************************************************************* doMenuChoice

void  doMenuChoice(SInt32 menuChoice)
{
  MenuID        menuID;
  MenuItemIndex menuItem;
  OSErr         osErr;
  MenuCommand   commandID;

  menuID   = HiWord(menuChoice);
```

```
  menuItem = LoWord(menuChoice);
  if(menuID == 0)
    return;
  else if(menuID == mFont)
    doFontMenu(menuItem);
  else
  {
    osErr = GetMenuItemCommandID(GetMenuRef(menuID),menuItem,&commandID);
    if(osErr == noErr && commandID != 0)
      doCommand(commandID);
  }

  HiliteMenu(0);
}

// ********************************************************************************* doCommand

void  doCommand(MenuCommand commandID)
{
  MenuRef menuRef;

  switch(commandID)
  {
    // ..................................................................................................................................................................... Apple/Application menu

    case 'abou':                                                                  // About
      drawItemString("\pAbout Menus2");
      break;

    // ..................................................................................................................................................................... File menu

    case 'quit':                                                                   // Quit
      gDone = true;
      break;

    // ..................................................................................................................................................................... Edit menu

    case 'undo':                                                                   // Undo
      drawItemString("\pUndo");
      break;

    case 'cut ':                                                                    // Cut
      drawItemString("\pCut");
      break;

    case 'copy':                                                                   // Copy
      drawItemString("\pCopy");
      break;

    case 'past':                                                                  // Paste
      drawItemString("\pPaste");
      break;

    case 'clea':                                                                  // Clear
      drawItemString("\pClear");
      break;
    // ..................................................................................................... Style ('xmnu') and Style (Programmatic) menu

    case 'plai':                                                                  // Plain
      gCurrentStyle = 0;
      doCheckStyleMenuItem(mStyleXmnu);
      doCheckStyleMenuItem(mStyleProg);
      break;

    case 'bold':                                                                   // Bold
      if(gCurrentStyle & bold)
        gCurrentStyle -= bold;
      else
        gCurrentStyle |= bold;
      doCheckStyleMenuItem(mStyleXmnu);
      doCheckStyleMenuItem(mStyleProg);
      break;

    case 'ital':                                                                 // Italics
      if(gCurrentStyle & italic)
        gCurrentStyle -= italic;
      else
      gCurrentStyle |= italic;
      doCheckStyleMenuItem(mStyleXmnu);
      doCheckStyleMenuItem(mStyleProg);
      break;

    case 'unde':                                                                // Underline
      if(gCurrentStyle & underline)
        gCurrentStyle -= underline;
      else
        gCurrentStyle |= underline;
      doCheckStyleMenuItem(mStyleXmnu);
      doCheckStyleMenuItem(mStyleProg);
```

```
      break;

    case 'outl':                                                    // Outline
      if(gCurrentStyle & outline)
        gCurrentStyle -= outline;
      else
        gCurrentStyle |= outline;
      doCheckStyleMenuItem(mStyleXmnu);
      doCheckStyleMenuItem(mStyleProg);
      break;

    case 'shad':                                                     // Shadow
      if(gCurrentStyle & shadow)
        gCurrentStyle -= shadow;
      else
        gCurrentStyle |= shadow;
      doCheckStyleMenuItem(mStyleXmnu);
      doCheckStyleMenuItem(mStyleProg);
      break;
    // ..................................................................................................................................................................... Size menu

    case 'ten ':                                                        // 10
      TextSize(10);
      doCheckSizeMenuItem(iTen);
      break;

    case 'twel':                                                        // 12
      TextSize(12);
      doCheckSizeMenuItem(iTwelve);
      break;

    case 'eigh':                                                        // 18
      TextSize(18);
      doCheckSizeMenuItem(iEighteen);
      break;

    case 'twen':                                                        // 24
      TextSize(24);
      doCheckSizeMenuItem(iTwentyFour);
      break;

    // ..................................................................................................................................................................... Special menu

    case 'firs':                                                      // First
      drawItemString("\pFirst Item");
      break;

    // ..................................................................................................................................................................... submenu

    case 'bat ':                                                        // Bat
      menuRef = GetMenuRef(mSubmenu);
      DisableMenuItem(menuRef,iBat);
      EnableMenuItem(menuRef,iBowl);
      drawItemString("\pBat");
      break;

    case 'bowl':                                                       // Bowl
      menuRef = GetMenuRef(mSubmenu);
      DisableMenuItem(menuRef,iBowl);
      EnableMenuItem(menuRef,iBat);
      drawItemString("\pBowl");
      break;

    case 'help':
      AHGotoPage(CFSTR("Menus Help"),CFSTR("Menus.htm"),NULL);
      break;
  }
}
// ********************************************************************************** doFontMenu

void  doFontMenu(MenuItemIndex menuItem)
{
  MenuRef      menuRef;
  OSStatus     osError;
  FMFontFamily currentFontFamilyReference;
  FMFontStyle  fontStyle;
  Str255       fontName;

  menuRef = GetMenuRef(mFont);

  osError = GetFontFamilyFromMenuSelection(menuRef,menuItem,&currentFontFamilyReference,
            &fontStyle);

  if(osError == noErr || osError == menuPropertyNotFoundErr)
  {
    TextFont(currentFontFamilyReference);

    CheckMenuItem(menuRef,gCurrentFontMenuItem,false);
    gCurrentFontMenuItem = menuItem;
```

```
    CheckMenuItem(menuRef,gCurrentFontMenuItem,true);

    GetMenuItemText(menuRef,menuItem,fontName);
    drawItemString(fontName);
  }
  else
    ExitToShell();
}

// ********************************************************************* doCheckStyleMenuItem

void  doCheckStyleMenuItem(MenuID menuID)
{
  MenuRef        styleMenuRef;
  static Boolean stringAlreadyDrawnOnce = false;

  styleMenuRef = GetMenuRef(menuID);

  CheckMenuItem(styleMenuRef,iPlain,    gCurrentStyle == 0);
  CheckMenuItem(styleMenuRef,iBold,      gCurrentStyle & bold);
  CheckMenuItem(styleMenuRef,iItalic,    gCurrentStyle & italic);
  CheckMenuItem(styleMenuRef,iUnderline,gCurrentStyle & underline);
  CheckMenuItem(styleMenuRef,iOutline,  gCurrentStyle & outline);
  CheckMenuItem(styleMenuRef,iShadow,    gCurrentStyle & shadow);

  TextFace(gCurrentStyle);

  if(!stringAlreadyDrawnOnce)
    drawItemString("\pStyle change");

  stringAlreadyDrawnOnce = !stringAlreadyDrawnOnce;
}
// ********************************************************************* doCheckSizeMenuItem

void  doCheckSizeMenuItem(MenuItemIndex menuItem)
{
  MenuRef sizeMenuRef;

  sizeMenuRef = GetMenuRef(mSize);

  CheckMenuItem(sizeMenuRef,gCurrentSizeMenuItem,false);
  CheckMenuItem(sizeMenuRef,menuItem,true);

  gCurrentSizeMenuItem = menuItem;

  drawItemString("\pSize change");
}

// ************************************************************************** drawItemString

void  drawItemString(Str255 eventString)
{
  RgnHandle tempRegion;
  WindowRef windowRef;
  Rect      scrollBox;

  windowRef = FrontWindow();
  tempRegion = NewRgn();

  GetWindowPortBounds(windowRef,&scrollBox);

  ScrollRect(&scrollBox,0,-30,tempRegion);
  DisposeRgn(tempRegion);

  MoveTo(8,286);
  DrawString(eventString);
}

// ********************************************************************************************
```

Demonstration Program Menus2 Comments

When this program is run, the user should choose Show Balloons from the Help menu and make menu choices from all menus, including the Apple menu. Choices should be made using the mouse and, where appropriate, the keyboard equivalents. The user should note:

- The extended modifier keys assigned to the last two items in the Style menus.
- The Command-key equivalents assigned to the items in the Size menu. (These are, in order, delete-to-the-left key, delete-to-the-right key, page-up key, and page-down key.)
- That the Font menu is WYSIWYG.
- That the marking character column has been deleted from the Special menu and the menu items in this menu are drawn in the Gadget font (assuming it is available).
- That the items in the submenu attached to the second item in the Special menu have color icons.
- The balloon help provided for all menus and menu items.

The Menus2 demonstration program package also includes a demonstration of Apple Help, including the methodology used to create an item in the Mac OS 8/9 Help menu. The Apple Guide file titled "Menus Guide", which will cause a "Menus Help" item to be created in the Mac OS 8/9 Help menu, should be retained in the same folder as the Menus2 application. An alias of the folder titled "Menus Help" should be placed in the Help folder in the System Folder (Mac OS 8/9) and in the user's Help folder (~/Library/Documentation/Help) (Mac OS X). You will then be able to access the help content by choosing Menus Help from the Help menu.

The help content does not provide user assistance for Menus2 programs as such. Rather, it provides a brief description of how to provide user assistance for your application using Apple Help.

Because this demonstration program is based on Menus1, the following comments exclude those for the functions that remain unchanged.

main

The calls to `RGBBackColor` and `RGBForeColor` set the window background and foreground colors to, respectively, dark blue and white.

doGetMenus

`doGetMenus` sets up the menu bar and the various menus.

`GetNewMBar` reads in the `'MENU'` resources for each menu specified in the `'MBAR'` resource and creates a menu object for each of those menus. (Note that the error handling here and in other areas of this program is somewhat rudimentary: the program simply terminates.) `SetMenuBar` makes the newly created menu list the current list.

The next block utilizes the Menu Manager function `CreateStandardFontMenu` in the creation of a non-hierarchical Font menu. Following the call to `CreateStandardFontMenu`, the process of making the menu WYSIWYG begins. The call to `CountMenuItems` returns the number of items in the menu. Then, for each of these items, `GetMenuItemText` gets the font's name, `GetFNum` gets the font number associated with the font name, and `SetMenuItemFontID` sets the font for the menu item. In the following if block, the current item is checkmarked if the item name equals the name of the small system font, and the global variable which keeps track of the currently selected font is assigned the item number.

The next block programmatically assigns extended modifier keys to the Outline and Shadow items in the Style (Programmatic) menu. The `SetMenuItemModifiers` calls assign Shift-Option-Control to the Outline item and Shift-Option to the Shadow item. (The extended modifier keys for the same two items in the Style (`'xmnu'`) menu are assigned in the associated `'xmnu'` resources.)

The next block inserts the application's single submenu into the submenu portion of the menu list and programmatically attaches it to the Special menu's second menu item. `GetNewMBar` does not read in the resource descriptions of submenus, so the first step is to read in the `'MENU'` resource with `GetMenu`. `InsertMenu` inserts a menu object for this menu into the menu list at the location specified in the second parameter to this call. (Using the constant `hierMenu` (-1) as the second parameter causes the menu to be installed in the submenu portion of the menu list.) The call to `GetMenuRef` gets a reference to the Special menu, which is used in the following call to `SetMenuHierarchicalID` to attach the submenu to the second item in the Special menu.

The following rather large block programmatically assigns command IDs to all items in the Style (Programmatic), Size, and Special menus and the submenu. (Command IDs for the File and Style (`'xmnu'`) menus are assigned in the associated `'xmnu'` resources. It is not possible to assign command IDs to the items in the Font menu.) The Command IDs are defined in the four-character-code format, which packs four one-byte characters together in a 32-bit value. For example, 'plai' expressed as hexadecimal is 0x706C6169. 70 is the ASCII code for p, 6C is the ASCII code for l, and 69 is the ASCII code for i.

The following block programmatically assigns a color icon to the second item in the submenu. The call to `GetCIcon` creates a `CIcon` data structure and initializes it from

data read in from the specified `'cicn'` resource. The handle to this structure is then passed as the last parameter in the `SetMenuItemIconHandle`, the third parameter specifying that the type of icon is a color icon. (The color icon for the first item in the submenu is assigned in the associated `'xmnu'` resource.)

The next block programmatically assigns command-key equivalents to the items of the Size menu. (Because the keys assigned are the two delete keys and the page-up and page-down keys, it is not possible to make these assignments within the `'MENU'` resource.) Also, a substitute glyph must be assigned, otherwise the correct glyphs will not be displayed. The calls to `SetItemCmd` assign the specified key to the menu item, and a substitute glyph is assigned via calls to `SetMenuItemGlyph`. If this is not done, the glyphs displayed will not be the correct visual representations of the keys. (These substitute glyphs could also have been specified in the keyboard glyph fields for these items in the menu's `'xmnu'` resource.)

In the next block, `SetMenuExcludesMarkColumn` is called to delete the marking character column from the Special menu and `SetMenuFont` is called to set the font for the menu items in this menu to Gadget (assuming that font is present).

In the next block, and only if the program is running on Mac OS X, `HMGetHelpMenu` is called to create a Help menu, `InsertMenuItem` is called to insert a single item in that menu, and `SetMenuItemCommandID` assigns a command ID to that item.

The next block sets checkmarks against the appropriate font, style and size menu items according to the initialized values of the associated global variables.

The call to `DrawMenuBar` draws the menu bar

Note that, in Carbon, the contents of the Apple Menu Items folder are automatically added to the Apple menu.

doMenuChoice

`doMenuChoice` extracts the menu ID and menu item number from the long integer returned by the `MenuSelect` and `MenuEvent` calls. An immediate return is made if the high word equals 0. The function "special cases" the Font menus, calling the function for handling choices from that menu. Otherwise, `GetMenuItemCommandID` is called. `GetMenuItemCommandID` returns zero as the function result if the call is successful, and a pointer to an integer representing the value of the item's command ID will be returned in the third parameter. If the call is successful, and if a zero is not returned in the third parameter, a command ID exists for the item. Accordingly, the command ID is passed in a call to the function `doCommand`.

`MenuSelect` and `MenuEvent` leave the menu title highlighted if an item was actually

chosen. Accordingly, the last line unhighlights the menu title when the action associated with the user's drop-down menu choice is complete.

doCommand

`doCommand` handles choices from those menus whose items have command IDs.

Note that the initial handling of all of the remaining menu items, regardless of which menu they belong to, is attended to within the one switch in the one function. The responses to the user choosing the various menu items is the same as in `Menus1`, except that the code relating to checkmarking the Style menu items has been added and the code for checkmarking the Size menu items and storing the current size has been divided between this function a further handling function (`doCheckSizeMenuItem`).

At the block titled Style (`'xmnu'`) and Style (Programmatic) menu, bits in the global variable `gCurrentStyle` are set or unset according to the font styles selected. The code reflects the fact that Bold, Italic, Underline, Outline and Shadow style selections are additive, not mutually exclusive, and that a selection of Plain must unset all bits in `gCurrentStyle`. The code also reflects the requirement that, except in the case of the Plain item, the selection of a checked item must cause that item to be unchecked, and vice versa. With `gCurrentStyle` set, the function `doCheckStyleMenuItem` is called to check/uncheck the relevant menu items as appropriate.

Note that the handling of the two submenu items has been changed to make the items mutually exclusive.

The 'help' command ID case applies only when the program is run on Mac OS X. The function `AHGoToPage` is called to deliver a request to load the specified HTML file in the specified Help book folder to the Help Viewer application.

doCheckStyleMenuItem

`doCheckStyleMenuItem` is called from `doMenuChoice` when an item in the Style menu is chosen. With the appropriate bit settings of `gCurrentStyle` attended to within `doMenuChoice`, a reference to the Style menu object is obtained. This is required for the six `CheckMenuItem` calls, which check or uncheck the individual menu items according to whether the third parameter evaluates to, respectively, true or false.

The call to `TextFace` sets the style for subsequent text drawing. The last line draws some text to prove that the desired effect was achieved.

Chapter 4. Introduction to Windows

IN THIS CHAPTER

- Scope of This Chapter
- Window Basics
- Types of Windows
- Window Regions
- Controls and Control Lists
- The Window List
- The Graphics Port and the Window Object
- Events in Windows
- Creating Your Application's Windows
- Managing Multiple Windows and Associated Data
- Handling Events
- Manipulating Windows
- Providing Help Balloons (Mac OS 8/9)
- Main Window Manager Constants, Data Types, and Functions
- Demonstration Program Windows1 Listing
- Demonstration Program Windows1 Comments

Scope of This Chapter

On Mac OS 8/9, Mac OS 8.5 introduced a number of significant new features and associated system software functions to the Window Manager. This chapter limits itself to the Window Manager as it existed prior to Mac OS 8.5, as influenced by the Carbon API. The additional features introduced with Mac OS 8.5 and Carbon are addressed in Chapter 16.

> **Note**
>
> By definition, all new functions introduced with Mac OS 8.5 and later are automatically supported by Carbon.

Window Basics

Windows, Documents, the Window Manager and Graphics Ports

A **window** is an area on the screen in which the user can enter or view information. Macintosh applications use windows for most communications and interactions with the user, including editing documents and presenting and acknowledging alerts and dialogs.

Users generally enter data in windows and your application typically lets the user save this data to a file. Your application typically creates **document windows** for this purpose. A document window is a view into the document. If the document is larger than the window, the window is a view of a portion of the document.

The Window Manager, which provides functions for managing windows, depends on QuickDraw. QuickDraw supports drawing into **graphics ports.** Each window represents a graphics port.

Standard Window Elements

> **Note**
>
> In the following, Mac OS 8/9 terminology is used. **Size box** equates to **resize control. Zoom box** equates to **zoom button. Close box** equates to **close button. Collapse box** equates to **minimize button.**

The user can manipulate windows using a set of standard window elements supported by the Window Manager. These standard elements are as follows:

- ***Title Bar.*** The title bar is the bar at the top of a window that displays the window's name, contains the close, zoom, and collapse boxes, and indicates whether a window is active. The name of a newly created window with which no document is yet associated should be "untitled". The name of a window containing a saved document should be the document's filename.

- ***Close Box.*** The close box allows the user to close a window. The close box is sometimes called the **go-away box.**

- ***Zoom Box.*** The zoom box allows the user to choose between two different window

sizes, one established by the user and one by the application. On Mac OS 8/9, the zoom box can be full, vertical, or horizontal.

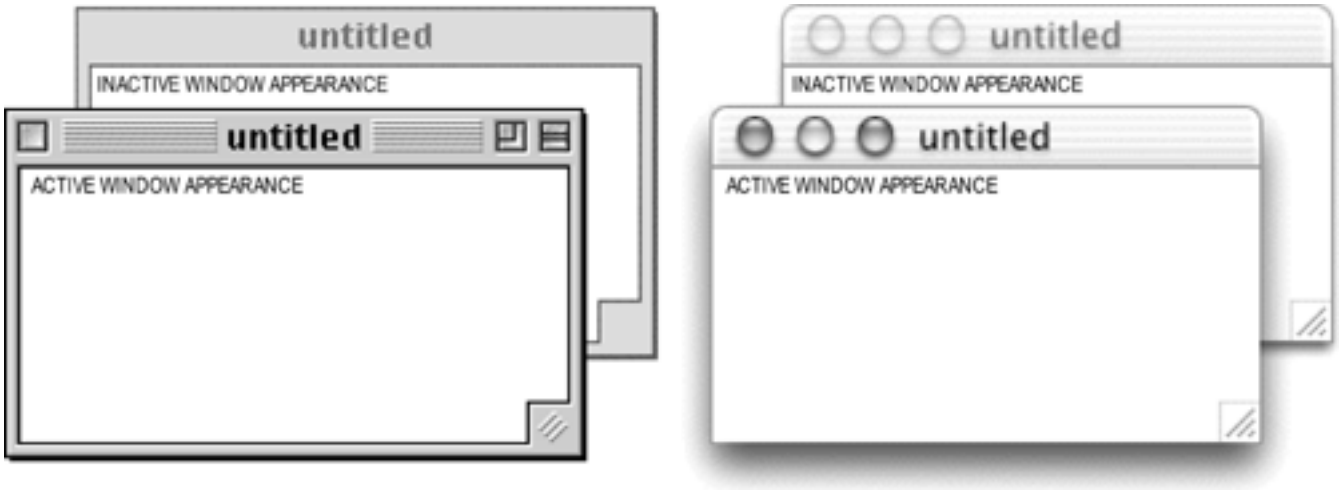
- ***Collapse Box.*** The collapse box allows the user to collapse (minimize on Mac OS X) and uncollapse a window.
- ***Size Box.*** The size box allows the user change the size of a window.
- ***Draggable Area.*** That part of the window's "frame," less the close, zoom, collapse, and size boxes.

Scroll bars, which allow the user to view different parts of a document containing more information than can be displayed in the window at the one time, are not an integral part of a window and must be separately created and managed. By convention, scroll bars are placed on the right and lower edges of those windows that require them.

Active and Inactive Windows

The **active window** is the window in which the user is currently working. It is identified by its general appearance (see Fig 1). All keyboard activity targets the active window and only the active window interacts with the user.

Figure 1. APPEARANCE OF ACTIVE AND INACTIVE WINDOWS



When the user activates one of your application's windows, the Window Manager redraws the window's title bar and frame/shadow, the close box/button, the title text, the zoom box/button, the collapse box/minimize button, and the size box/resize control as shown in Fig 1. Your application must reinstate the appearance of the rest of the window to its state prior to the deactivation, activating any controls, drawing the scroll box (scroller, in Mac OS X terminology) in the same position, restoring the insertion point, and highlighting the previous selection, and so on.

When a window belonging to your application becomes **inactive,** the Window Manager redraws the title bar and frame/shadow, as shown in Fig 1. Your application must deactivate any controls, remove highlighting from selections, and so on.

When the user clicks in an inactive document window, your application should make the window active but should not make any selections in response to the click. To make a selection, the user should be required to click again. This behavior protects the user from unintentionally losing an existing selection when activating the window.

Window Layering

On Mac OS 8/9, all of an application's document windows are in the one layer. On Mac OS X, document windows from different applications can be interleaved, and clicking on a window to bring it forward does not affect the layering of other windows. That said, if an application has a Window menu, choosing Bring All To Front will cause all of the application's document windows to be brought in front of all of the document windows of all other applications. Clicking on the application's icon in the Dock has the same effect.

Types of Windows

The Window Manager defines a large number of window **types,** which may be classified as follows:

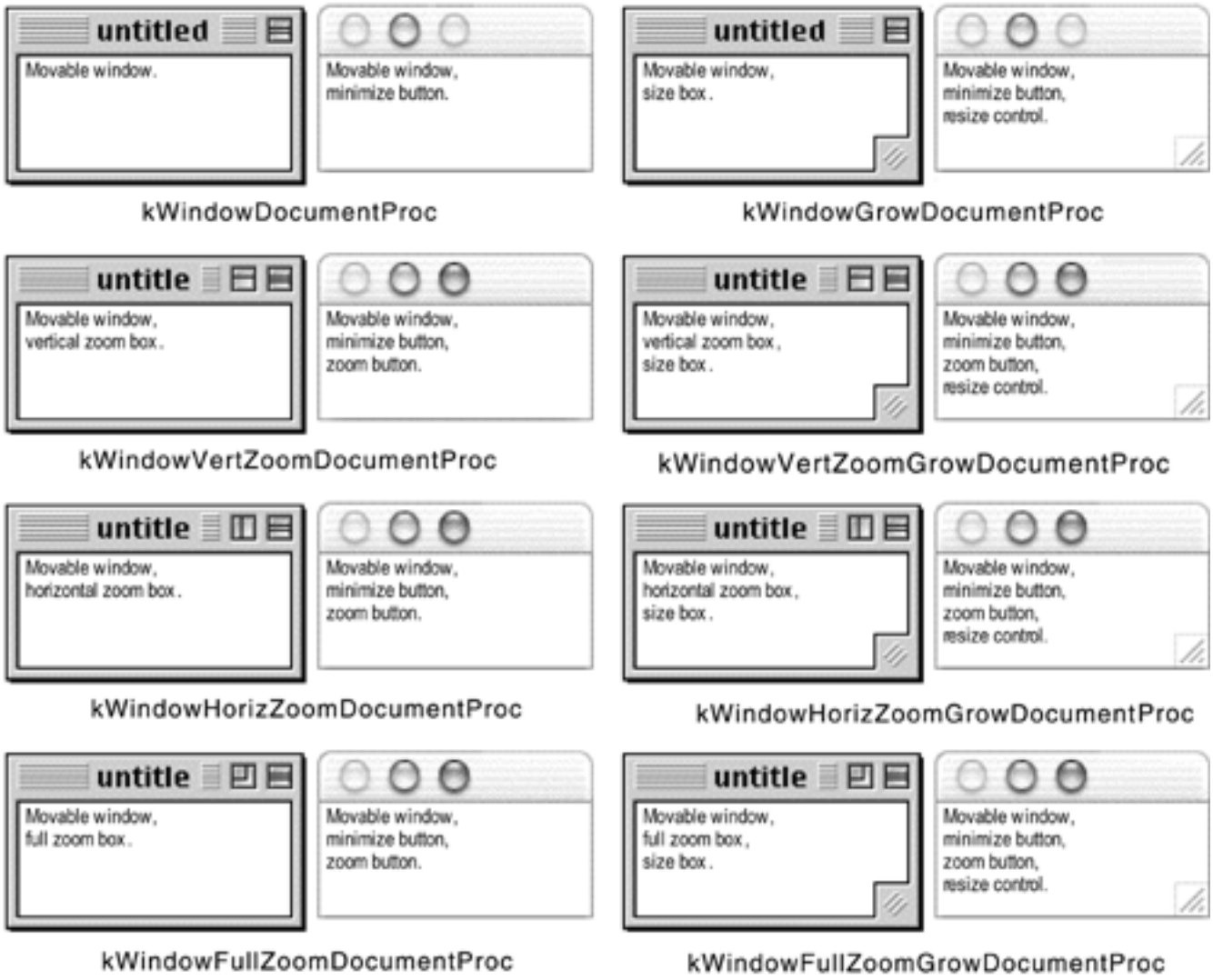
- Document types
- Dialog and alert types
- Floating window types

Window types are often referred to by the constant used in `'WIND'` resources, and by certain Window Manager functions, to specify the type of window required. That constant determines both the visual appearance of the window and its behavior.

Document Types

Fig 2 shows the eight available window types for documents and the constants that represent those types.

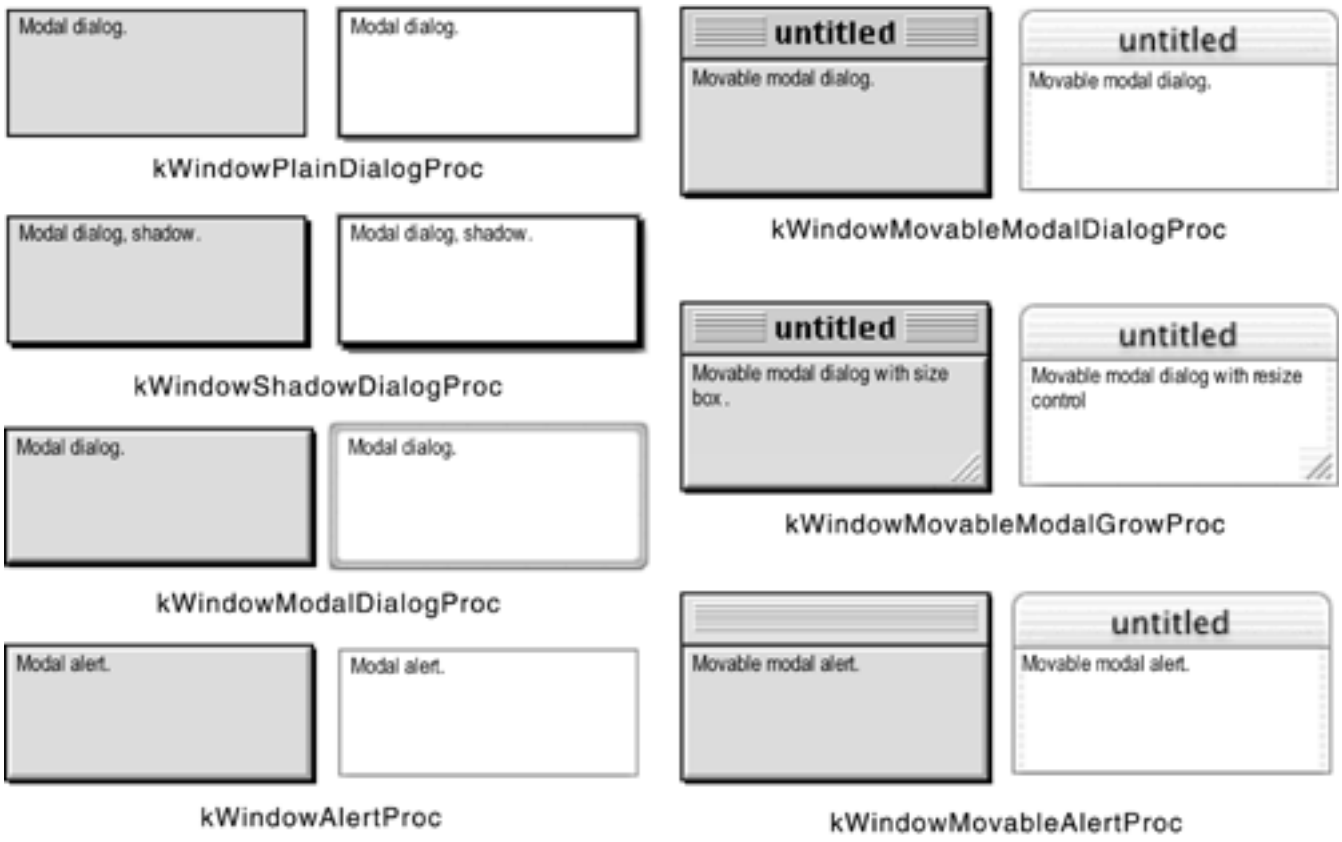
Figure 2. WINDOW TYPES FOR DOCUMENTS



Dialog and Alert Types

Fig 3 shows the seven available window types for modal and movable modal dialogs and alerts and the constants that represent those types. (The document window type represented by the constant `kWindowDocumentProc` is used for modeless dialogs.)

Figure 3. WINDOW TYPES FOR DIALOGS AND ALERTS



Floating Window Types

Figs 4 and 5 show the sixteen available window types for floating windows and the constants that represent those types.

Figure 4. WINDOW TYPES FOR FLOATING WINDOWS (TITLE BAR AT TOP)



Figure 5. WINDOW TYPES FOR FLOATING WINDOWS (PSUEDO TITLEBAR AT SIDE)


Floating window, side title bar.
Floating window, side title bar.
kWindowFloatSideProc
Floating window, side title bar, size box.
Floating window, side title bar, resize control.
kWindowFloatSideGrowProc
Floating window, side title bar, vertical zoom box.
Floating window, side title bar, zoom button.
kWindowFloatSideVertZoomProc
Floating window, side title bar, vertical zoom box, size box.
Floating window, side title bar, zoom button, resize control.
kWindowFloatSideVertZoomGrowProc
Floating window, side title bar, horizontal zoom box.
Floating window, side title bar, zoom button.
kWindowFloatSideHorizZoomProc
Floating window, side title bar, horizontal zoom box, size box.
Floating window, side title bar, zoom button, resize control.
kWindowFloatSideHorizZoomGrowProc
Floating window, side title bar, full zoom box.
Floating window, side title bar, zoom button.
kWindowFloatSideFullZoomProc
Floating window side title bar, full zoom box, size box.
Floating window, side title bar, zoom button, resize control.
kWindowFloatSideFullZoomGrowProc

Window Definition IDs

The constants shown in Figs 2, 3, 4, and 5 each represent a specific **window definition ID.** A window definition ID is a 16-bit value which contains the resource ID of the window's **window definition function** in the upper 12 bits and a **variation code** in the lower 4 bits:

- ***Window Definition Function.*** The system software and various Window Manager functions call a window's window definition function (WDEF) when they need to perform certain window-related actions, such as drawing or re-sizing a window's frame.

- ***Variation Code.*** A single WDEF can support up to 16 different window types. The WDEF defines a variation code-an integer from 0 to 15-for each window type it supports.

Four WDEFs (resource IDs 64, 65, 66, and 67) are associated with the three classifications of window types.

The window definition ID is derived by multiplying the resource ID of the WDEF by 16 and adding the variation code to the result, as is shown in the following:

WDEF Resource ID	***Variation Code***	***Window Definition ID(Value)***	***Window Definition ID (Constant)***

64	0	64 * 16 + 0 = 1024	kWindowDocumentProc
64	1	64 * 16 + 1 = 1025	kWindowGrowDocumentProc
64	2	64 * 16 + 2 = 1026	kWindowVertZoomDocumentProc
64	3	64 * 16 + 3 = 1027	kWindowVertZoomGrowDocumentProc
64	4	64 * 16 + 4 = 1028	kWindowHorizZoomDocumentProc
64	5	64 * 16 + 5 = 1029	kWindowHorizZoomGrowDocumentProc
64	6	64 * 16 + 6 = 1030	kWindowFullZoomDocumentProc
64	7	64 * 16 + 7 = 1031	kWindowFullZoomGrowDocumentProc
65	0	65 * 16 + 0 = 1040	kWindowPlainDialogProc
65	1	65 * 16 + 1 = 1041	kWindowShadowDialogProc
65	2	65 * 16 + 2 = 1042	kWindowModalDialogProc
65	3	65 * 16 + 3 = 1043	kWindowMovableModalDialogProc

65	4	65 * 16 + 4 = 1044	kWindowAlertProc
65	5	65 * 16 + 5 = 1045	kWindowMovableAlertProc
65	6	65 * 16 + 6 = 1046	kWindowMovableModalGrowProc
66	1	66 * 16 + 1 = 1057	kWindowFloatProc
66	3	66 * 16 + 3 = 1059	kWindowFloatGrowProc
66	5	66 * 16 + 5 = 1061	kWindowFloatVertZoomProc
66	7	66 * 16 + 7 = 1063	kWindowFloatVertZoomGrowProc
66	9	66 * 16 + 9 = 1065	kWindowFloatHorizZoomProc
66	11	66 * 16 + 11 = 1067	kWindowFloatHorizZoomGrowProc
66	13	66 * 16 + 13 = 1069	kWindowFloatFullZoomProc
66	15	66 * 16 + 15 = 1071	kWindowFloatFullZoomGrowProc
67	1	67 * 16 + 1 = 1073	kWindowFloatSideProc

67	3	67 * 16 + 3 = 1075	kWindowFloatSideGrowProc
67	5	67 * 16 + 5 = 1077	kWindowFloatSideVertZoomProc
67	7	67 * 16 + 7 = 1079	kWindowFloatSideVertZoomGrowProc
67	9	67 * 16 + 9 = 1081	kWindowFloatSideHorizZoomProc
67	11	67 * 16 + 11 = 1083	kWindowFloatSideHorizZoomGrowProc
67	13	67 * 16 + 13 = 1085	kWindowFloatSideFullZoomProc
67	15	67 * 16 + 15 = 1087	kWindowFloatSideFullZoomGrowProc

Window Type Usage

For Documents

A `kWindowFullZoomGrowDocumentProc` window is normally used for document windows because it supports all window manipulation elements. Note that because you can optionally suppress the close box/button when you create the window, the Window Manager does not necessarily draw/highlight that particular element. Also note that when the related document contains more data than will fit in the window, you must add scroll bars.

For Modal Alerts and Modal Dialogs

Modal alerts and modal dialogs are simply special-purpose windows. Modal alerts generally use the window type `kWindowAlertProc` and modal dialogs generally use window type `kWindowModalDialogProc`.

Note

The creation and handling of alerts and dialogs is addressed in detail in Chapter 8.

For Movable Modal Alerts and Movable Modal Dialogs

Movable modal alerts and movable modal dialogs are used when you want the user to be able to move the alert or dialog window or to bring another application to the foreground before the dialog is dismissed. Movable modal alerts use the window type `kWindowMovableAlertProc` and movable modal dialogs use the window type `kWindowMovableModalDialogProc`.

For Modeless Dialogs

Modeless dialogs allow the user to perform other tasks within the application without first dismissing the dialog. User interface guidelines require that the `kWindowDocumentProc` window type, which can be moved or closed but not resized or zoomed, be used for modeless dialogs.

Window Regions

The Window Manager recognizes the special-purpose **regions** shown in Figs 6 and 7.

Figure 6. WINDOW REGIONS-MAC OS 8/9

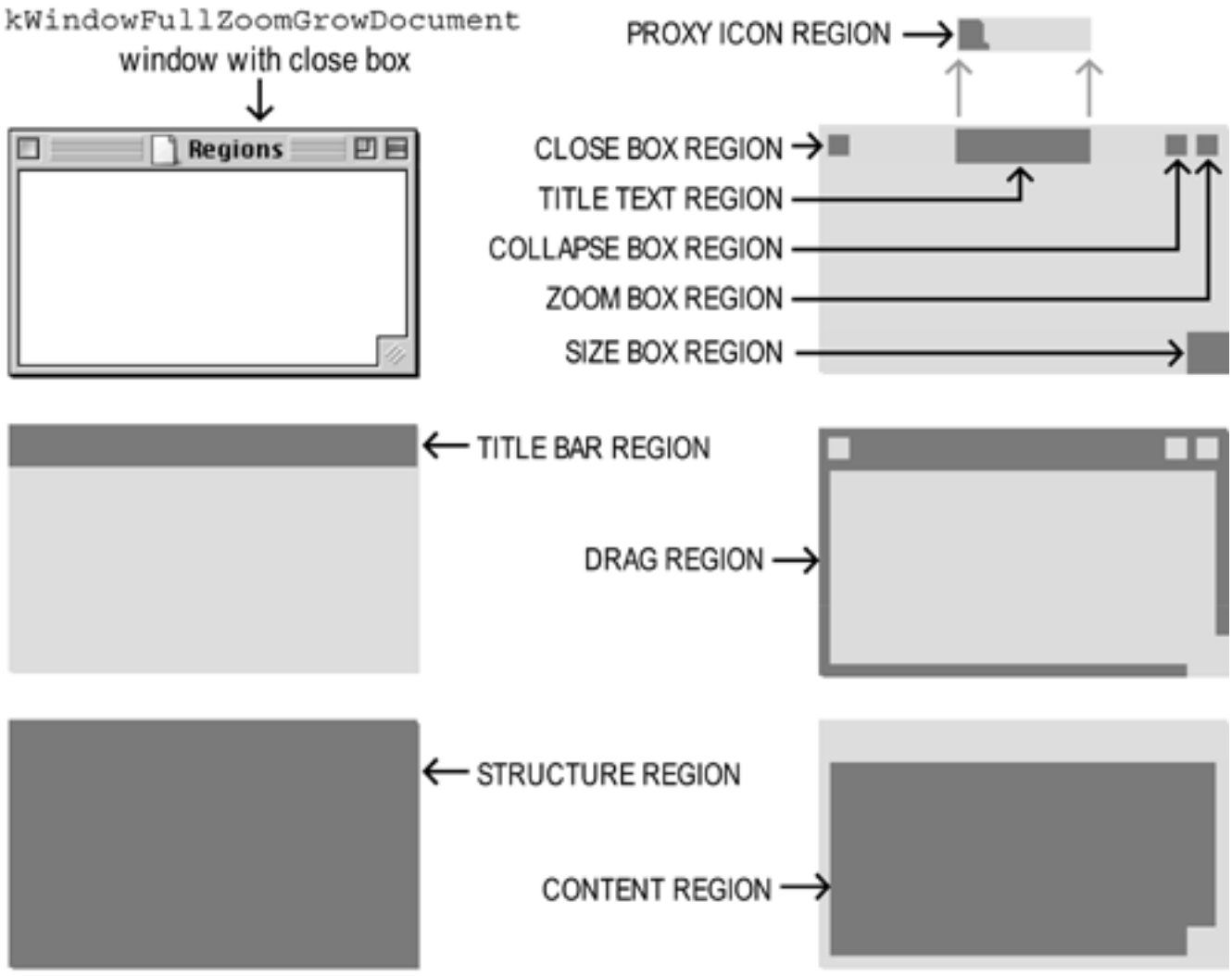
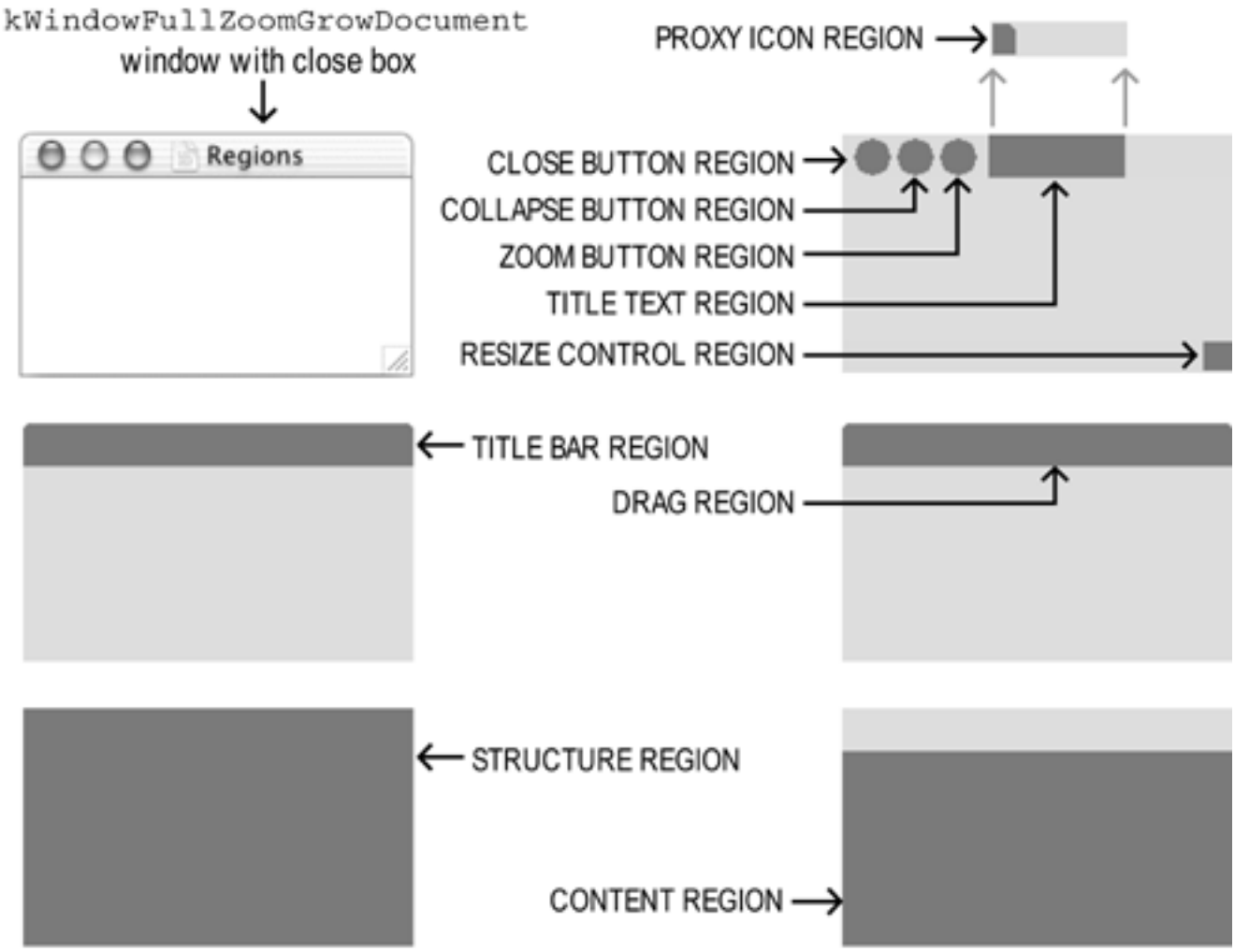


Figure 7. WINDOW REGIONS-MAC OS X



Note

A region is an arbitrary area, or set of areas, on the QuickDraw coordinate plane. The outline of a region is one or more closed loops. Regions are explained in more detail in Chapter 12.

Handles to these and two other window-related regions, which are represented by constants of type `RegionWindowCode`, may be obtained via a call to `GetWindowRegion`. The definitions of these regions, and the constants which represent them, are as follows:

Region	*Constant*	*Definition*
Title bar region	`kWindowTitleBarRgn`	The entire area occupied by a window's title bar, including the title text region.

Title text region	`kWindowTitleTextRgn`	That portion of a window's title bar that is occupied by the name of the window and the window proxy icon. (See Chapter 16.)
Close box/button region	`kWindowCloseBoxRgn`	The area occupied by a window's close box/button.
Zoom box/button region	`kWindowZoomBoxRgn`	The area occupied by a window's zoom box/button.
Drag region	`kWindowDragRgn`	On Mac OS X, this equates to the title bar region. On Mac OS 8/9, this includes the window frame, including the title bar and window outline, but excluding the close box, zoom box, collapse box, and size box (if any).
Size box/resize control region	`kWindowGrowRgn`	The area occupied by a window's size box/resize control.
Collapse box/minimize button region	`kWindowCollapseBoxRgn`	The area occupied by a window's collapse box/ minimize button.
Proxy icon region	`kWindowTitleProxyIconRgn`	The area occupied by the window proxy icon. (See Chapter 16.)
Structure region	`kWindowStructureRgn`	The entire area occupied by a window, including the frame and content region. (The window may be partially off-screen but its structure region does not change.)
Content region	`kWindowContentRgn`	That part of a window in which the contents of a document and the window's controls (including scroll bars) are displayed.

Update region	`kWindowUpdateRgn`	Contains all areas of a window's content region that need updating (re-drawing).
Global port region	`kWindowGlobalPortRgn`	The bounds of the window's graphics port, in global coordinates, even when the window is collapsed.

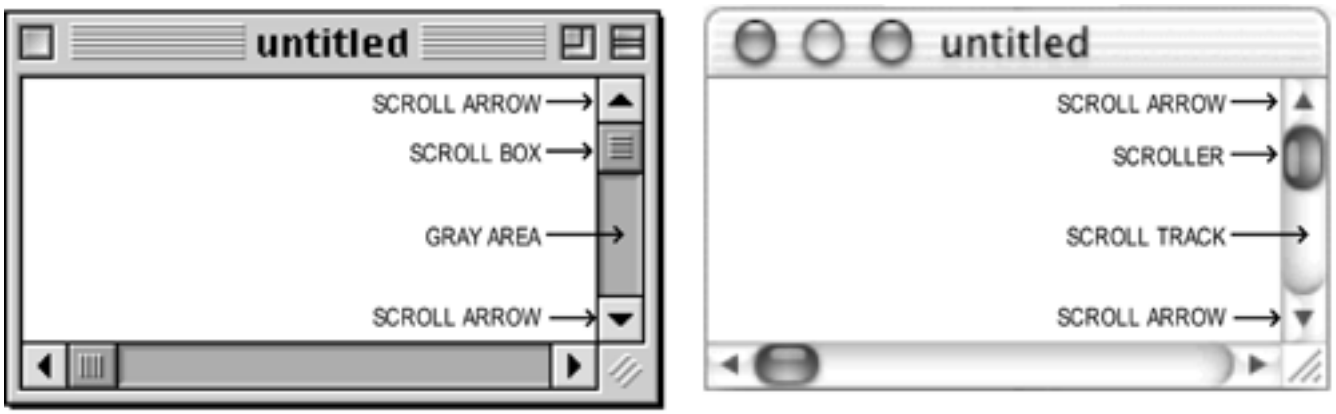
The Desktop (Gray) Region

Another region of some relevance to the Window Manager is the **desktop region** (sometimes known as the **gray region**). The desktop region is the region below the menu bar, including all screen real estate in a system equipped with multiple monitors. The Window Manager maintains a pointer to the desktop region in a low-memory global variable named `GrayRgn`. You can get a handle to the desktop region using the function `GetGrayRgn`.

Controls and Control Lists

Windows may contain **controls.** The most common control in a window is the **scroll bar** (see Fig 8), which should be included in the window when there is more data than can be shown at one time in the space available. The Control Manager is used to create, display and manipulate scroll bars.

Figure 8. SCROLL BARS



All controls "belonging" to an individual window and are displayed within the graphics port that represents that window. Entries pointing to the descriptions of a window's controls are stored in a **control list** maintained for that window by the Window Manager.

The Window List

At any one time, many windows, from many applications, may be displayed on the desktop. To track all of the windows on the desktop, the Window Manager maintains a private data structure called the **window list.** The arrangement of the entries in this list reflects the current front-to-back ordering of the windows on the desktop, the frontmost (active) window being the first in the list.

The function `GetWindowList` returns a reference to the window object (see below) for the first window in the window list. The function `GetNextWindow` returns a reference to the window object for the next window in the window list.

The Graphics Port and the Window Object

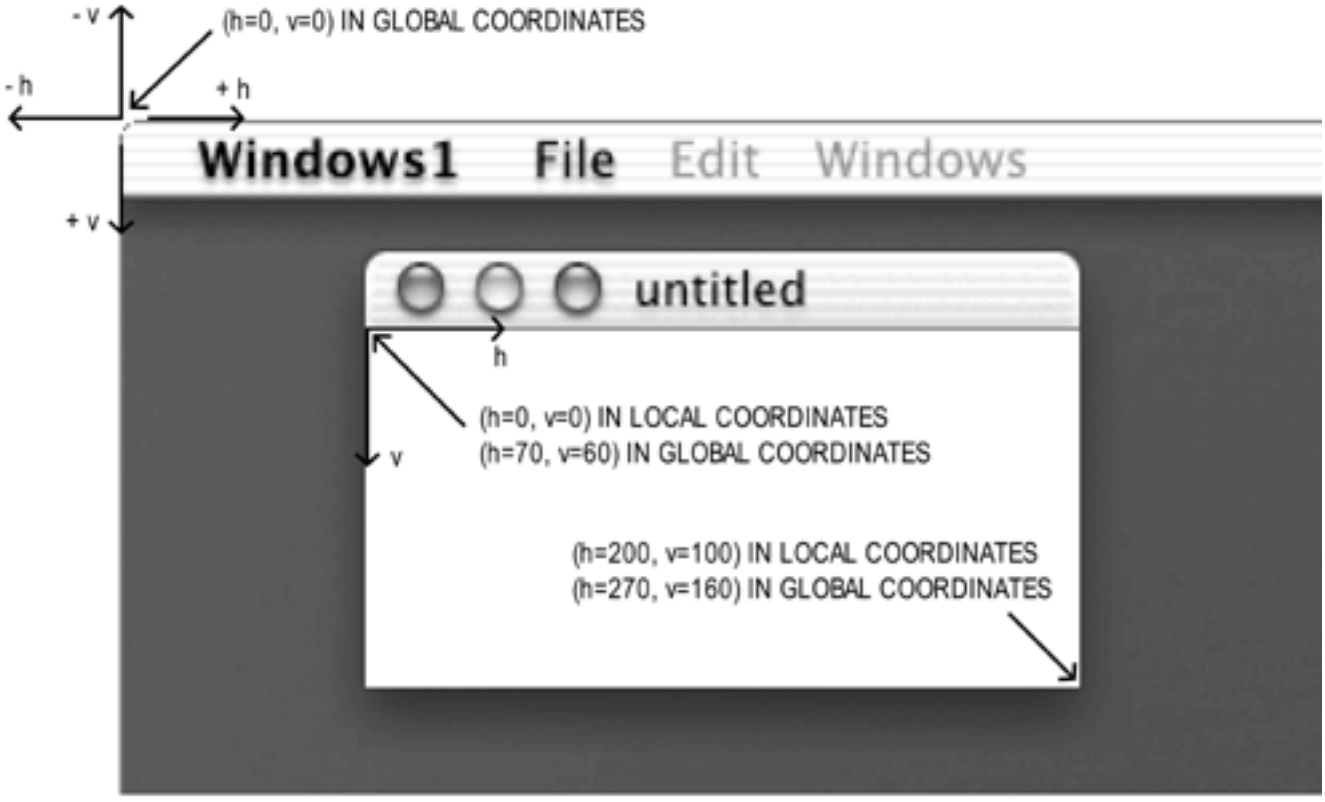
The Graphics Port

Each window represents a QuickDraw graphics port, an opaque object of type `CGrafPort` which describes a drawing environment with its own coordinate system. The Window Manager creates a graphics port when it creates the window.

> **Note**
>
> The graphics port is addressed in detail in Chapter 11.

The location of a window on the screen is defined in QuickDraw's **global coordinates.** QuickDraw's global coordinates originate at the top left corner of the main screen and extend vertically and horizontally within the range -32768 to 32,767. The positive x-axis extends rightward and the positive y-axis extends downward (see Fig 9).

Figure 9. A WINDOW'S LOCAL AND GLOBAL COORDINATE SYSTEMS



The graphics port object stores a rectangle called the **port rectangle.** In a graphics port representing a window, the port rectangle represents the window's content region. Within the port rectangle, the drawing area is described in **local coordinates.** Fig 9 illustrates the local and global coordinate systems for a window which is 100 pixels high by 200 pixels wide, and which is placed with its content region 70 pixels down and 60 pixels to the right of the upper-left corner of the screen.

When the Window Manager creates a window, it places the origin of the local coordinate system at the upper-left corner of the window's port rectangle. Note, however, that the Event Manager describes mouse events in global coordinates, and that you must do most of your window manipulation in global coordinates.

The Window Object

The Window Manager stores information about individual windows in opaque data structures called **window objects.** The data type `WindowPtr` is defined as a pointer to a window object:

```
typedef struct OpaqueWindowPtr* WindowPtr;
```

Note that the data type `WindowPtr` is equivalent to the newer data type `WindowRef`:

```
typedef WindowPtr WindowRef;
```

Carbon Note

One of the fundamental differences between the Classic API and the Carbon API is that, in the Classic API, the data type `WindowPtr` is defined as a pointer to a graphics port structure, not to a window structure. The first field in the Classic API's window structure is a graphics port structure, meaning that the graphics port structure has the same address as the window structure in which it resides. In the Classic API, a `WindowPtr` must be cast to a `WindowPeek` (which is defined as a pointer to a window structure) in order for the fields of the window structure to be directly accessed.

Accessor Functions

Accessor functions are provided to access the information in window objects. The accessor functions are as follows:

Function	*Description*

`GetWindowPort`	Gets a pointer to the specified window's graphics port object.
`GetWindowKind`	Gets the window kind (dialog or application) of the specified window.
`SetWindowKind`	Sets the window kind (dialog or application) of the specified window.
`IsWindowVisible`	Determines whether the specified window is visible.
`ShowWindow`	Makes the specified window visible.
`HideWindow`	Makes the specified window invisible.
`ShowHide`	Makes the specified window visible or invisible without affecting front-to-back ordering.
`IsWindowHilited`	Determines whether the specified window is highlighted.
`HiliteWindow`	Highlights or unhighlights a window.
`GetWindowRegion`	Gets a handle to the specified window's structure, content, and update regions. **Note:** Handles to the other regions shown in Figs 6 and 7 are not included in the window object. The WDEF determines the location of those particular regions.
`GetWTitle`	Gets the specified window's title.
`SetWTitle`	Sets the specified window's title.
`GetWindowPic`	Gets a handle to the picture stored in the window object by `SetWindowPic`.
`SetWindowPic`	Stores a handle to a picture in the window object, causing the Window Manager to draw the picture instead of generating an update event.
`GetWRefCon`	Gets the reference constant stored in the specified window's window object by `SetWRefCon`.
`SetWRefCon`	Sets a reference constant in the specified window's window object.
`GetWindowStandardState`	Gets the window's standard state rectangle.
`SetWindowStandardState`	Sets the window's standard state rectangle.

`GetWindowUserState`	Gets the window's user state rectangle.
`SetWindowUserState`	Sets the window's user state rectangle.

Events in Windows

As stated in Chapter 2, the Window Manager itself generates two types of events central to window management, namely, activate events and update events.

One of the Window Manager's main tasks is to report the location of the cursor when the application receives a mouse-down event. As was also stated in Chapter 2, the Window Manager function `FindWindow` may be used to determine whether the cursor is in a window when the mouse-down occurs and, if it is in a window, in exactly which window and which part of that window. `FindWindow` is thus the function which enables you to distinguish between those mouse-down events that affect the window itself and those that affect the document displayed in the window.

Creating Your Application's Windows

You typically create document and floating windows from resources of type `'WIND'`, although you can create them programmatically using the function `NewCWindow`. Additional methods of creating windows, including floating windows, programmatically are described in Chapter 16.

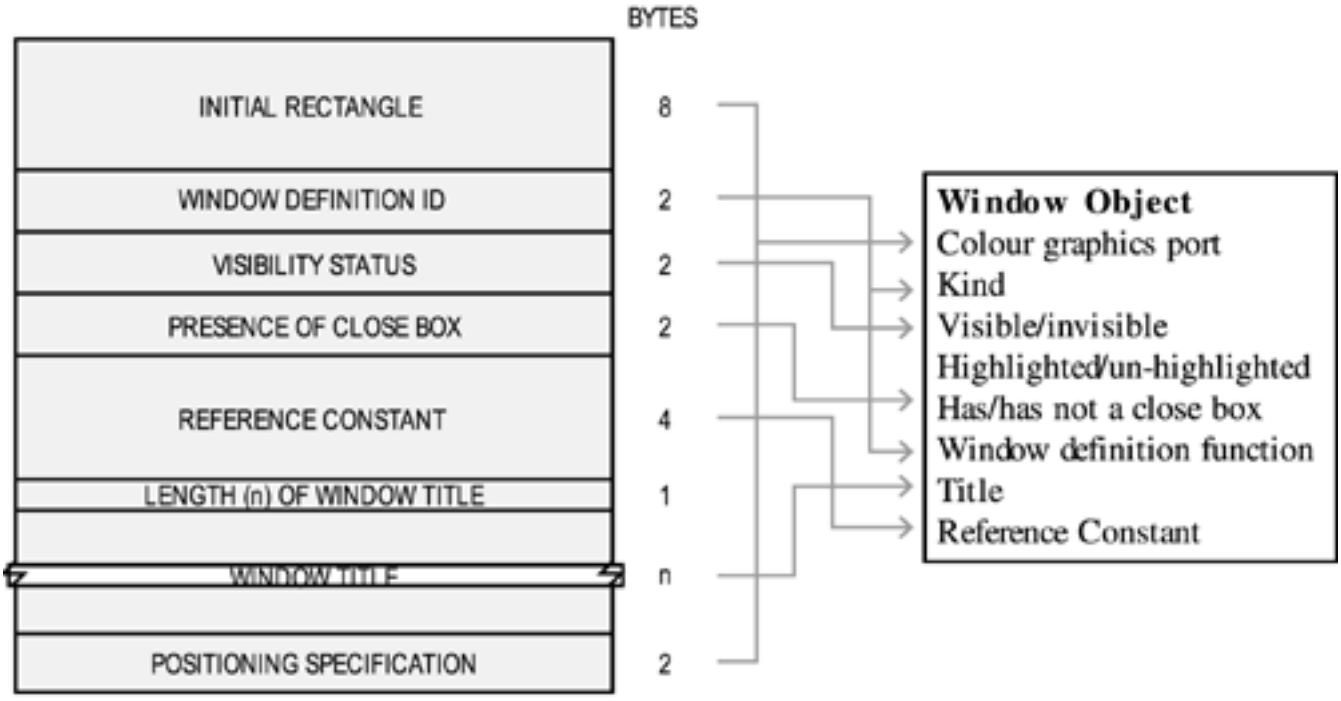
'WIND' Resources

When creating resources with Resorcerer, it is advisable that you refer to a diagram and description of the structure of the resource and relate that to the various items in the Resorcerer editing windows. Accordingly, the following describes the structure of the resource associated with the creation of document and floating windows.

Structure of a Compiled 'WIND' Resource

Fig 10 shows the structure of a compiled `'WIND'` resource and how it "feeds" the window object.

Figure 10. STRUCTURE OF A COMPILED WINDOW ('WIND') RESOURCE



The following describes the main fields of the `'WIND'` resource:

Field	*Description*
INITIAL RECTANGLE	A rectangle that defines the initial size and placement, in global coordinates, of the window's content region. This rectangle can be changed before displaying the window, either programmatically or by using an optional positioning specification.
WINDOW DEFINITION ID	The window's definition ID.
VISIBILITY STATUS	Specifies whether the window is to be visible or invisible. Note that this really means whether the Window Manager displays the window; it does not necessarily mean that the window can be seen on the screen. (A visible window might be completely covered by other windows; nevertheless its visibility status is still "visible.")
PRESENCE OF CLOSE BOX/ BUTTON	Specifies whether the window is to have a close box/button.
REFERENCE CONSTANT	A reference constant that your application can use for whatever data it needs to associate with the window. When it builds a new window object, the Window Manager stores in that object whatever value you specify in this field. (You can also store a reference constant in the window object programmatically via a call to `SetWRefCon`.)
WINDOW TITLE	A string that specifies the window's title.
POSITIONING SPECIFICATION	A positioning specification (optional). If this field contains a positioning specification, it overrides the window position established by the rectangle in the first field. The positioning constants which may be assigned to this field are very useful for specifying the initial position of dialogs, alerts, and windows for new documents. However, the position (and size) of a new window intended to display a previously saved document should be the same as the window position (and size) when the document was last displayed.

Positioning Specification

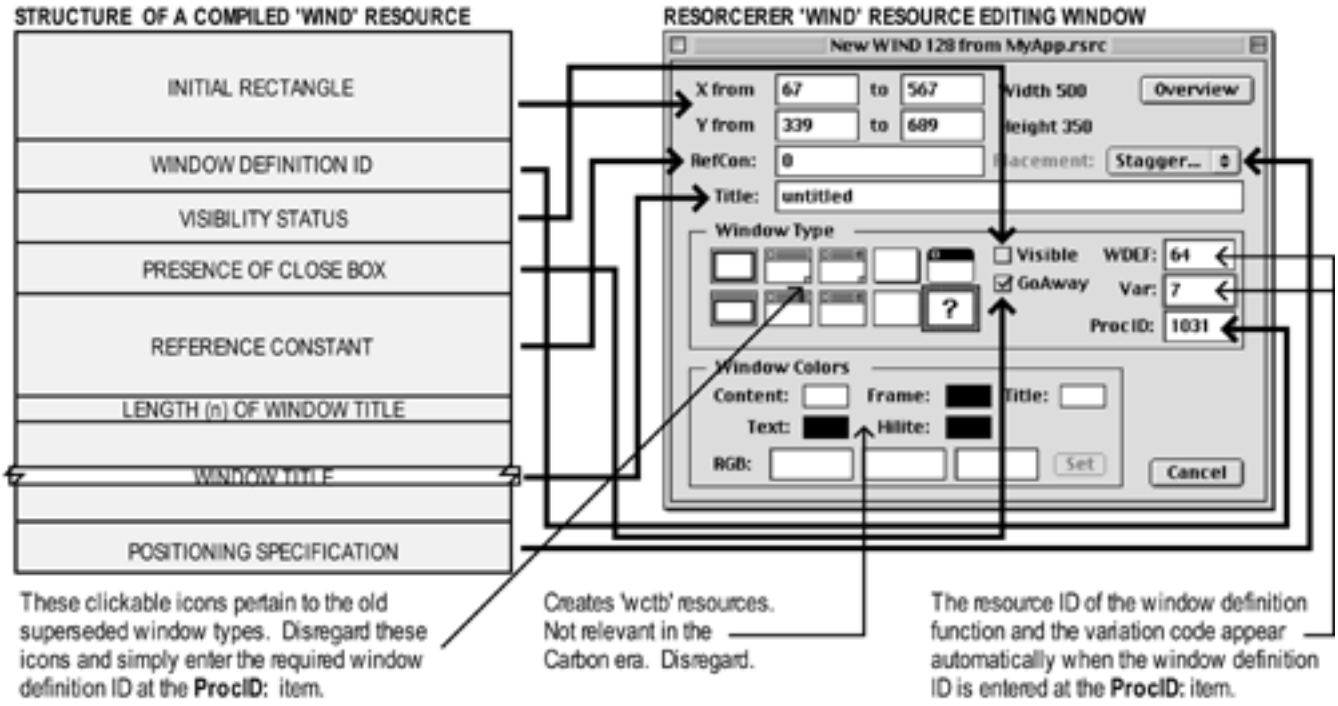
The constants for the positioning specification field are as follows:

Constant	*Value*	*Meaning*
`kWindowNoPosition`	`0x0000`	Use initial rectangle.
`kWindowDefaultPosition`	`0x0000`	Use initial rectangle.
`kWindowCenterMainScreen`	`0x280A`	Center on main screen.
`kWindowAlertPositionMainScreen`	`0x300A`	Place in alert position on main screen.
`kWindowStaggerMainScreen`	`0x380A`	Stagger on main screen.
`kWindowCenterParentWindow`	`0xA80A`	Center on parent window.
`kWindowAlertPositionParentWindow`	`0xB00A`	Place in alert position on parent window.
`kWindowStaggerParentWindow`	`0xB80A`	Stagger relative to parent window.
`kWindowCenterParentWindowScreen`	`0x680A`	Center on parent window screen.
`kWindowAlertPositionParentWindowScreen`	`0x700A`	Alert position on parent window screen.
`kWindowStaggerParentWindowScreen`	`0x780A`	Stagger on parent window screen.

Creating a 'WIND' Resource Using Resorcerer

Fig 11 shows a `'WIND'` resource being created with Resorcerer.

Figure 11. CREATING A 'WIND' RESOURCE USING RESORCERER



Creating the Window from the 'WIND' Resource

`GetNewCWindow` is used to create a window from a `'WIND'` resource.

Adding Scroll Bars

If a window requires scroll bars, you typically create them from `'CNTL'` resources at the time that you create the document window, and then display them when you make the window visible (see Chapter 7).

Window Visibility

If the `'WIND'` resource specifies that the new window is visible, `GetNewCWindow` displays the window immediately. If you are creating a document window, however, it is best to create the window in an invisible state and then make it visible when you are ready to display it. The right time to display a window depends on whether the window is associated with a new or a saved document.

- If you are creating a window because the user is creating a new document, you should display the window immediately (by calling `ShowWindow`).
- If you are creating a new window to display a saved document, you should retrieve the document and draw it in the window before calling `ShowWindow`.

Positioning a New Document Window on the Desktop

The positioning constants previously described allow you to position new windows

automatically. When used, those positioning constants concerned with staggering new window placement will ensure that the Window Manager will use any vacated position for the next new window.

Getting Available Window Positioning Bounds

Carbon introduced the function `GetAvailableWindowPositioningBounds`, which allows your application to determine the available window positioning bounds on a given screen. The function returns the bounds of the screen rectangle less the menu bar and, on Mac OS X, the Dock.

Positioning a Saved Document Window on the Desktop

For windows created for the purpose of displaying a saved document, you should replicate the size and location of the document's window as it was when the document was last saved. When the user saves a document, you must therefore also save the **user state** rectangle and the current **zoom state** of the window (that is, whether the window is in the user state or the **standard state**).

User State, Standard State, and Zoom State

Some explanation of user state and standard state is necessary. The user state is the last size and position the user, through sizing and dragging actions, established for a window. The standard state is the size and position that your application defines as being best for the display of the data contained in the window.

The user and standard states are stored in the window object and may be set and retrieved using the functions `GetWindowStandardState`, `SetWindowStandardState`, `GetWindowUserState`, and `SetWindowUserState` (see the section "Accessor Functions").

In addition, the function `IsWindowInStandardState` allows your application to determine the current zoom state-that is, whether the window is zoomed "out" to the standard state or zoomed "in" to the user state.

Saving the Window State

Returning to the matter of saving the user state and the current zoom state of the window, for windows with zoom boxes/buttons you typically store this data as a custom resource in the resource fork of the document file.

Drawing a Window's Contents

Your application is responsible for drawing a window's contents. It typically uses the Control Manager to draw the window's controls and then draws the user data itself.

Managing Multiple Windows and Associated Data

Your application is likely to have multiple windows open on the desktop at once, each of which will have some form of data, such as text, associated with it. This data, which is external to the window object, is regarded as the "property" of an individual window.

As previously stated, you can store a reference constant in a window object using the function `SetWRefCon`. Typically, your application will use `SetWRefCon` to store a handle to a structure containing the window's external data. This structure, usually referred to as a **document structure,** might hold a handle to the text being edited, handles to the scroll bars, a file reference number and a file system specification for the document's file, plus a flag indicating whether data has changed since the last save, as shown in the following example:

```
typedef struct
{
  TEHandle      editRec;
  ControlHandle vScrollBar;
  Controlhandle hScrollbar;
  short         fileRefNum;
  FSSpec        fileFSSpec;
  boolean       windowTouched;
}  docStructure;
typedef docStructure **docStructureHdl;
```

Your application can retrieve the reference constant using the function `GetWRefCon`.

Handling Events

Handling Mouse Events

Your application, when it is the active application, receives all mouse-down events in its menu bar and its windows. When it receives a mouse-down event, your application should call `FindWindow` to ascertain which window, and which part of that window, the mouse-down occurred in. (In this context, the menu bar is considered to be a window part). The application should then take the appropriate action based on which window, and which part of that window, the mouse-down occurred in.

Mouse-Downs in Inactive Windows

When you receive a mouse-down event in an inactive document window or modeless dialog, and if the active window is a document window or a modeless dialog, you should call `SelectWindow`, passing it the window reference. `SelectWindow` re-layers the windows as necessary, removes highlighting from the previously active window, brings the newly activated window to the front, highlights it and generates the activate and update events necessary to tell all affected applications which windows must be redrawn.

Note that, if the active window is a modal or movable modal alert or dialog, no action is required by your application. Modal and movable modal alerts and dialogs are handled by the `ModalDialog` function, which does not pass the event to your application.

Handling Keyboard Events

Whenever your application is the active application, it receives all key-down events (except, of course, for the system-defined Command-Shift-number key sequences).

When you receive a key-down event, you should first check whether the user is holding down a modifier key and another key at the same time. Your application should respond to key-down events by inserting data into the document, changing the display or taking other appropriate actions. Typically, your application provides feedback for standard keystrokes by drawing the character on the screen.

Handling Update Events

Handling Update Events-Mac OS 8/9

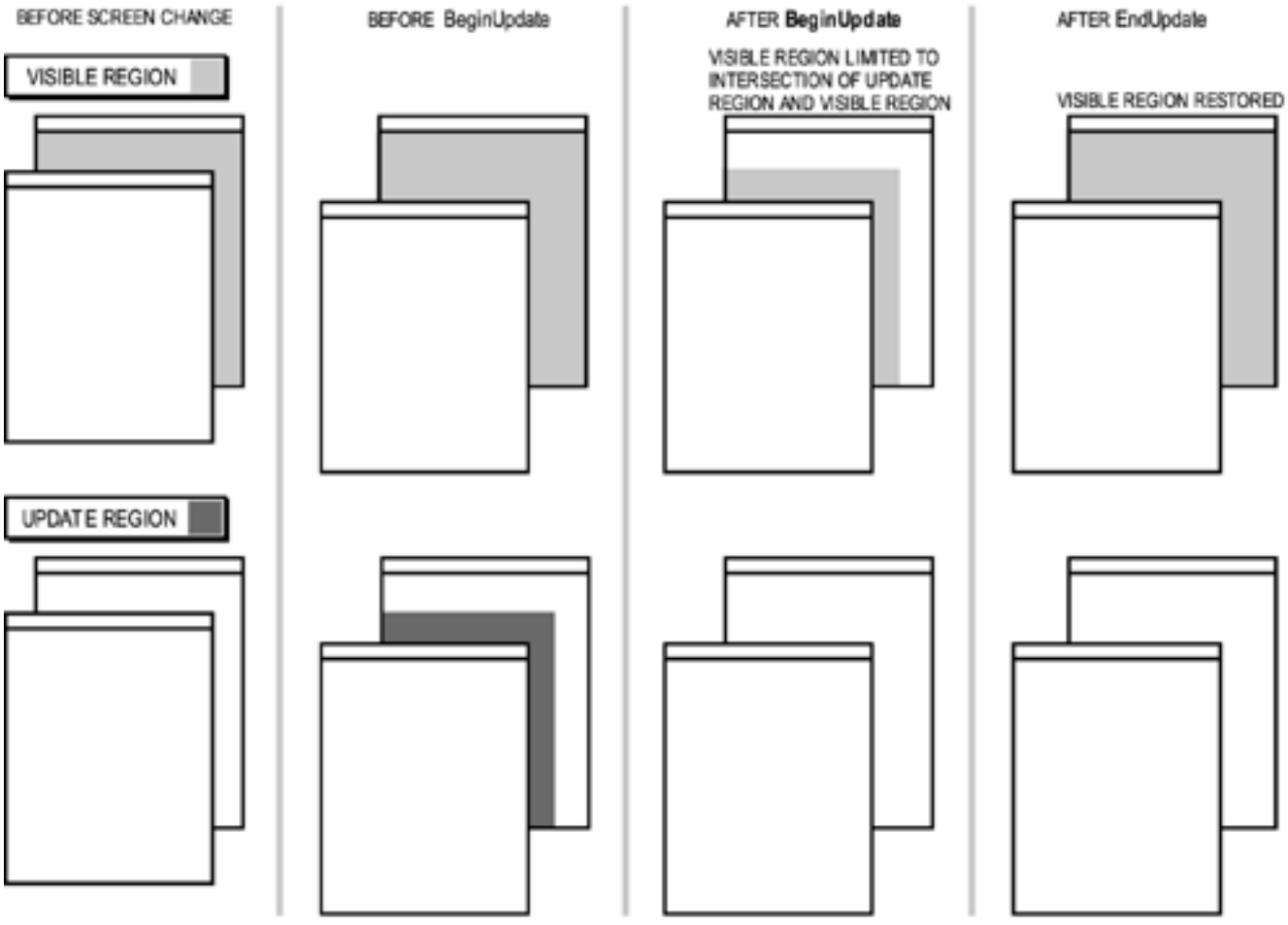
On Mac OS 8/9, the Window Manager maintains an update region for each window. The

update region represents the parts of a window's content region that have been affected by changes to the desktop and need redrawing. The Event Manager continually scans the window objects of all the windows on the desktop, looking for non-empty update regions. If it finds an update region that is not empty, it generates an update event for that window.

When your application receives an update event, it should redraw the content area. When your application redraws the content area, the Window Manager clips all screen drawing to the **visible region** of the window's graphics port. The visible region is that part of a graphics port that is not covered by other windows-that is, the part that is actually visible on screen. The Window Manager stores a handle to the visible region in the graphics port object.

Before redrawing the content area, your application should call `BeginUpdate` and, when it has completed the drawing, it should call `EndUpdate`. As shown in Fig 12, `BeginUpdate` temporarily adjusts the visible region to equate to the intersection of the visible region and the update region. Because QuickDraw limits its drawing to this temporarily modified visible region, only those parts of the window that actually need updating are drawn. `BeginUpdate` also clears the update region, thus ensuring that the Event Manager does not continue sending an endless stream of update events.

Figure 12. EFFECTS OF BeginUpdate AND EndUpdate ON VISIBLE AND UPDATE REGIONS



When the drawing is completed, as shown in Fig 12, `EndUpdate` restores the visible region of the graphics port to the full visible region.

The reason for these update region/visible region machinations is that the handle to the update region is stored in the window object while the handle to the visible region is stored in the graphics port object. QuickDraw knows the graphics port object intimately, but knows nothing about the window object or its contents. QuickDraw needs something it can work with, hence the above process whereby the visible region is temporarily made the equivalent of the update region while QuickDraw does its drawing.

Manipulating the Update Region

Your application can force or suppress update events by manipulating the update region. You can call `InValWindowRect` or `InvalWindowRgn` to add a rectangle or region to the update region, thus causing an update event to be generated and, as a consequence, that area to be redrawn. You can also remove a rectangle or region from the update region by calling `ValidWindowRect` or `ValidWindowRgn` so as to decrease the time spent redrawing. For example, an unaffected text area could be removed from the update region of a window that is being resized.

Handling Update Events-Mac OS X

On Mac OS X, windows are double-buffered, meaning that your application does not draw into the window's graphics port itself but rather into a separate buffer. The Window Manager flushes the buffer to the window's graphics port when your application calls `WaitNextEvent`. On Mac OS X, your application does not require update events to cater for the situation where part, or all, of a window's content region has just been exposed as a result of the user moving an overlaying window.

The receipt of an update event on Mac OS X simply means that your application should draw the required contents of the window. The swapping of visible and update regions required on Mac OS 8/9 is not required, so calls to `BeginUpdate` and `EndUpdate` are irrelevant (and ignored) on Mac OS X.

As is the case on Mac OS 8/9, your application can force the generation of an update event by calling `InValWindowRect` or `InvalWindowRgn`. (Your application can also force the buffer to be flushed to the window's graphics port by calling the QuickDraw function `QDFlushPortBuffer`. This is required, for example, when your application is drawing periodically in a loop that does not call `WaitNextEvent`.)

Type-Dependent Update Functions

An update function should typically first determine whether the type of window being updated is a document window or some other type of window. If the window is a document window, a

document window updating function should be called. If the window is a modeless dialog, an updating function for that modeless dialog should be called.

Handling Activate Events

Activate events are generated by the Window Manager to inform your application that a window is becoming active or is about to be made inactive. Each activate event specifies the window to be changed and whether the window is to be activated or deactivated.

Your application typically triggers activate events itself by calling `SelectWindow` following a mouse-down event in a nonactive window. `SelectWindow` brings the window in which the mouse-down occurred to the front, adds highlighting to that window, and removes highlighting from the previously active window. `SelectWindow` then generates one activate event to tell your application to perform its part of the deactivation of the previously active window, followed by and another activate event to tell your application to perform its part of the activation of the newly activated window.

When your application receives the event for the window about to be made inactive, it should hide the window's controls and remove any highlighting of selections. When your application receives the event for the newly activated window, it should draw the controls and restore the content area as necessary. This latter might involve, for example, adding the insertion point at its former location or highlighting the previously highlighted selection.

The function for handling activate events should typically first determine whether the window being activated/deactivated is a document window or a modeless dialog. It should then perform the appropriate activation/deactivation actions. The function does not need to check for modal alerts or modal dialogs because the Dialog Manager's `ModalDialog` function automatically handles activate events for those windows.

Manipulating Windows

Dragging a Window

When a mouse-down event occurs in the title bar, your application should call `DragWindow`, which tracks the user's actions until the mouse button is released. `DragWindow` moves an image of the window on the screen as the user moves the mouse. When the user releases the mouse, `DragWindow` redraws the window in its new location. The window is then activated (unless the Command key was pressed during the drag).

The area within which the window may be dragged is specified by a `Rect` variable passed in `DragWindow`'s `boundsRect` parameter.

Carbon Note

In Carbon, assigning `NULL` to the `boundsRect` parameter has the effect of setting the parameter to the bounding rectangle of the desktop region.

Zooming a Window

The zoom box/button allows the user to alternate between two window sizes and positions known as the user state and the standard state. To amplify the previous description of user state and standard state:

- The user state is the window size and location established by the user. (If the user has not yet moved or resized the window, the user state is the size and location of the window when it was created.)
- The standard state is the size and position that your application specifies as being the optimum for the display of the data contained in the window. In a word-processing program, for example, a window in the standard state might show the full width of a page and as much length as will fit on the screen. If the user changes the page size using the Page Setup dialog, the application might adjust the standard state width to reflect the new page width.
- The user and standard states are stored in the window object. The Window Manager sets the initial standard and user states when it fills in the window object on creation, and it updates the user state whenever the user resizes the window.

Mac OS 8.5 introduced new functions for implementing window zooming. Prior to Mac OS 8.5, when `FindWindow` returned either `inZoomIn` or `inZoomOut`, your application

would call `TrackBox` to handle highlighting of the zoom box/button and to determine whether the cursor was inside or outside the zoom box/button when the button was released. If `TrackBox` returned `true`, your application would call `ZoomWindow` to resize the window.

The advantage of the `ZoomWindowIdeal` function introduced with Mac OS 8.5 is that, unlike `ZoomWindow`, it zooms the window in accordance with the following Apple human interface guidelines relating to a window's standard state:

- "A window should move as little as possible when zooming between the user state and standard state, to avoid distracting the user."
- "A window in its standard state should be positioned so that it is entirely on one screen."
- "If a window straddles more than one screen in the user state, when it is zoomed to the standard state it should be zoomed to the screen that contains the largest portion of the window's content region."
- "If the ideal size for the standard state is larger than the destination screen, the dimensions of the standard state should be that of the destination screen, minus a few pixels of boundary. If the destination screen is the main screen, space should also be left for the menu bar."
- "When a window is zoomed from the user state to the standard state, the top left corner of the window should remain anchored in place; however, if the standard state of the window cannot fit on the screen with the top left corner anchored, the window should be "nudged" so that the parts of the window in the standard state that would fall offscreen are, instead, just onscreen."

Other advantages of `ZoomWindowIdeal` are that

- It allows you to specify, in its `ioIdealSize` parameter, the desired height and width of the window's content region in the standard state.
- On Mac OS X, it takes account of the current height of the Dock when zooming to the standard state and constrains the zoom accordingly.

When `ZoomWindowIdeal` is used, another function introduced with Mac OS 8.5 (`IsWindowInStandardState`) must be used to determine the appropriate part code (`inZoomIn` or `InZoomOut`) to pass in `ZoomWindowIdeal`'s `partCode` parameter.

Vertical or Horizontal Zoom Boxes-Mac OS 8/9

Your application should ensure that, when a vertical zoom box is clicked, only the vertical size of the associated window changes. Similarly, when a horizontal zoom box is clicked, your application should ensure that only the horizontal size of the associated window changes.

Resizing a Window

Mac OS 8.5 introduced a new function for implementing window re-sizing. Prior to Mac OS 8.5, when the user pressed the mouse button in the size box, your application would call `GrowWindow`. This function displayed a **grow image,** a dotted outline of the window frame and scroll bar area which expanded and contracted as the user dragged the size box. When the user released the mouse button, `GrowWindow` returned a long integer which described the window's new height (in the high-order word) and width (in the low-order word). A value of zero indicated that the window size did not change. When the mouse-button was released and `GrowWindow` returned a non-zero value, your application called `SizeWindow` to draw the window in its new size.

The function introduced with Mac OS 8.5 is `ResizeWindow`. `ResizeWindow` moves a grow image around the screen, following the user's cursor movements, and handles all user interaction until the mouse button is released. Unlike the function `GrowWindow`, there is no need to follow this call with a call to `SizeWindow`. Once resizing is complete, `ResizeWindow` draws the window in its new size.

If you pass `NULL` in `ResizeWindow`'s `sizeConstraints` parameter, resizing will be constrained by the default minimum size (64 by 64 pixels) and the default maximum size (the bounding rectangle of the desktop region). However, the option is available to supply custom upper and lower resizing limits in the fields of a `Rect` variable passed in the `sizeConstraints` parameter. The limits represented by each field are as follows:

- `sizeConstraints.top` represents the minimum vertical measurement.
- `sizeConstraints.left` represents the minimum horizontal measurement.
- `sizeConstraints.bottom` represents the maximum vertical measurement.
- `sizeConstraints.right` represents the maximum horizontal measurement.

Note that the values assigned represent window dimensions, *not* screen coordinates.

Following a window resize, your application should adjust its scroll bars and window contents to accommodate the new size.

Carbon Note

In Carbon, `NULL` may be assigned to `ResizeWindow`'s `newContentRect` parameter if the new dimension of the window's content region is not required.

Closing a Window

The user closes a window by clicking in the close box/button or by choosing Close from the File menu.

When the user clicks in the close box/button, `TrackGoAway` should be called to track the mouse until the user releases the mouse button. If `TrackGoAway` returns `true`, meaning that the user did not release the mouse button outside the close box/button, your application should call its function for closing down the window.

The actions taken by your window closing function depend on the type of information the window contains and whether the contents need to be saved. The function should cater for different types of windows, that is, modeless dialogs (which may be merely hidden with `HideWindow` rather than closed completely) and standard document windows. In the latter case, the function should check whether any changes have been made to the document since it was opened and, if so, provide the user with an opportunity to save the document to a file before closing the window. (This whole process is explained in detail at Chapter 18.)

As for the window itself, `DisposeWindow` removes a window from the screen, removes it from the window list, and discards all of its data storage, including the window object.

Hiding and Showing a Window

Ordinarily, when the user elects to close a window, you dispose of the window. In some circumstances, however, it may be more appropriate to simply hide the window instead of removing its data structures. It is usually more appropriate, for example, to hide, rather than dispose of, modeless dialogs. That way, when the user next chooses the relevant menu command, the dialog is already available, and in the same location, as when it was last used.

`HideWindow` hides a window. `ShowWindow` will make the window visible and `SelectWindow` will make it the active window.

Providing Help Balloons (Mac OS 8/9)

Help Balloons-'hrct' and 'hwin' Resources

The Mac OS 8/9 system software provides help balloons for the title bar, draggable area, close box, zoom box, and collapse box for windows created with the standard WDEFs. Where applicable, you should provide help balloons for the content area of your windows.

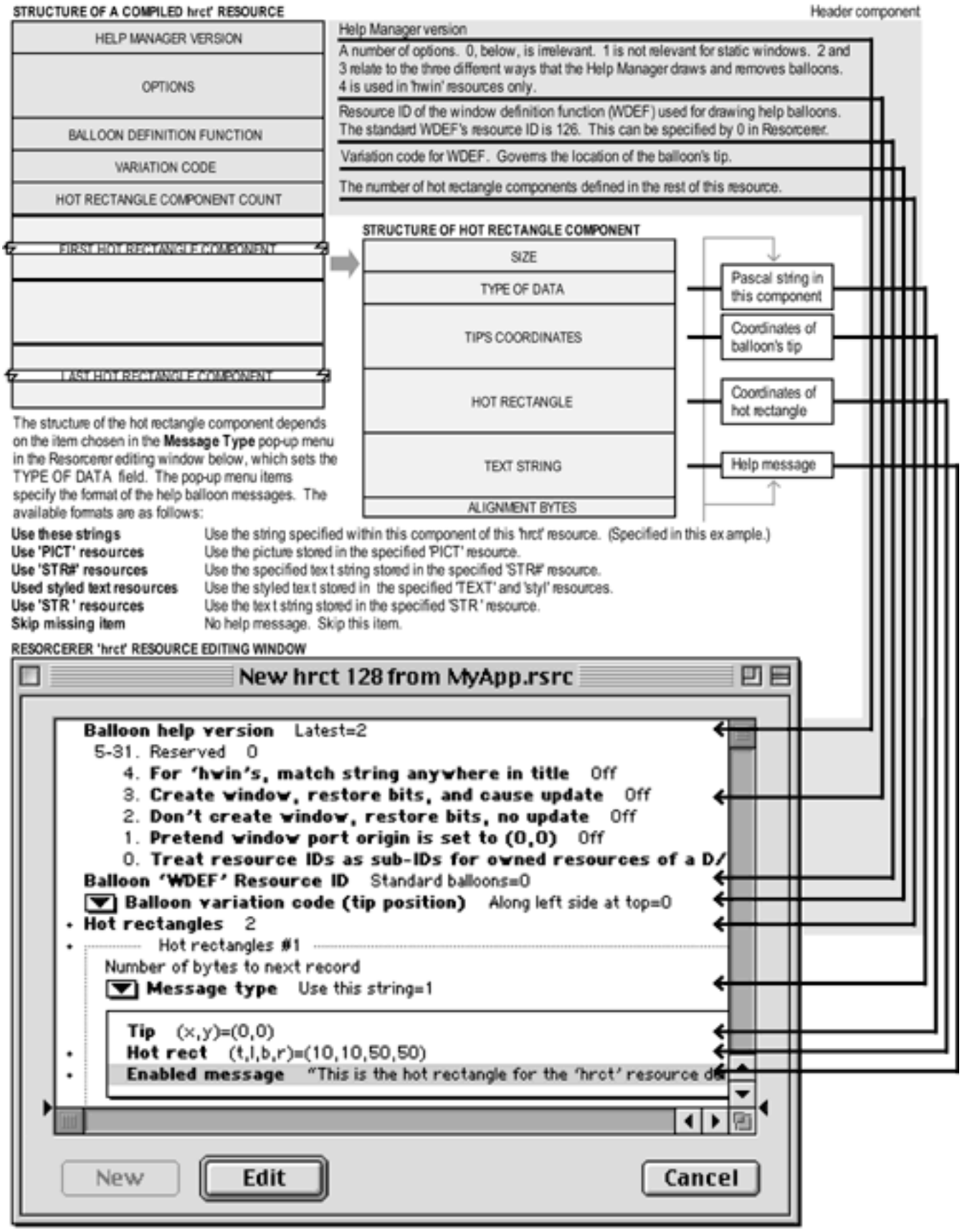
How you choose to provide help balloons for the content area depends mainly on whether your windows are **static** or **dynamic.** A static window does not change its title or reposition any of the objects within its content area. A dynamic window can reposition any of it objects within its content area, or its title may change. For example, any window that scrolls past areas of interest to the user is a dynamic window because the object with associated help balloons can change location as the user scrolls. The following addresses the case of static windows only.

Help balloons for static document and floating windows are defined in `'hrct'` and `'hwin'` resources.

Creating 'hrct' and 'hwin' Resources Using Resorcerer

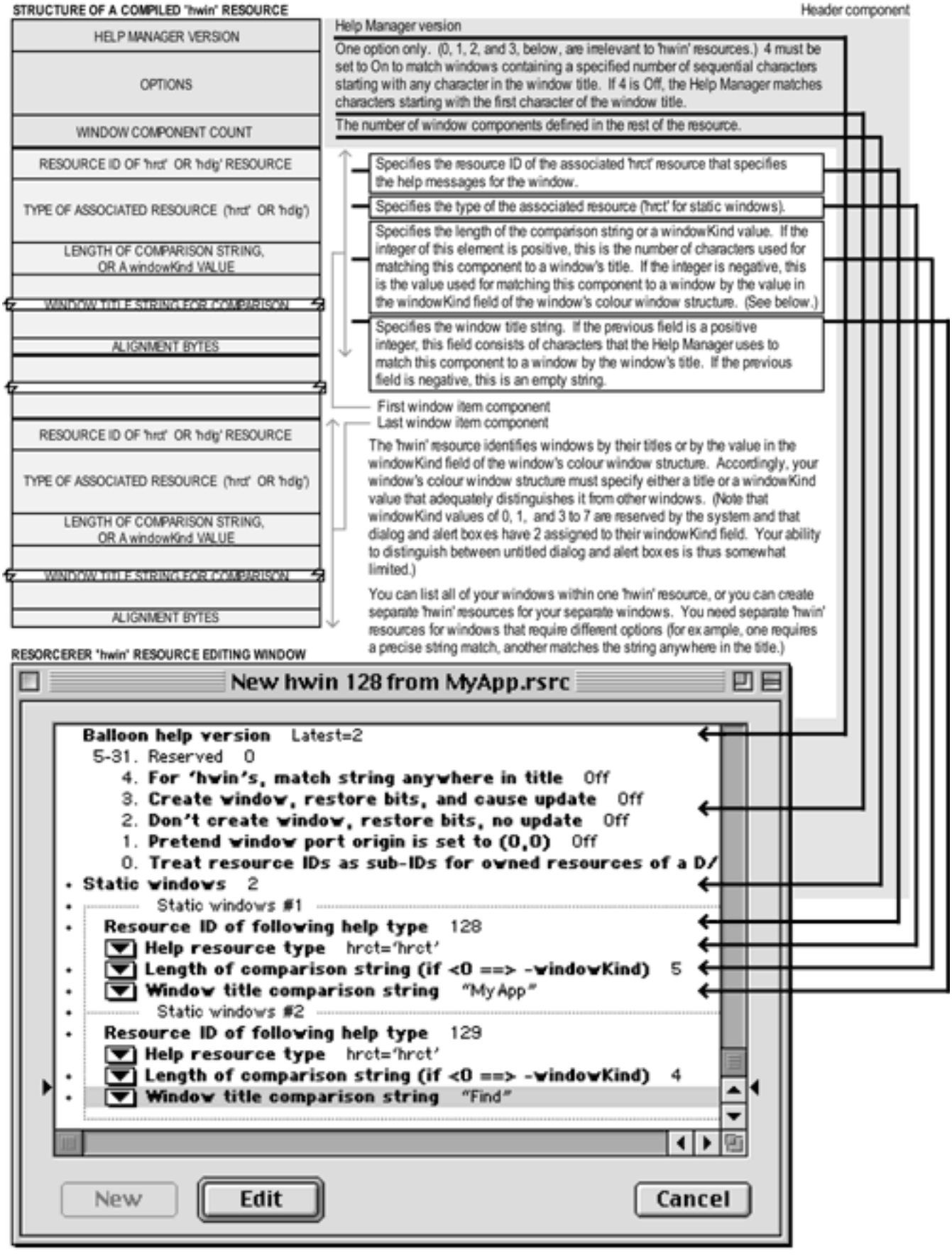
The `'hrct'` (rectangle help) resource is used to define **hot rectangles** for displaying help balloons in a static window and to specify the help messages for those balloons. All `'hrct'` resources must have resource IDs equal to or greater than 128. Fig 13 shows an `'hrct'` resource being created using Resorcerer.

Figure 13. CREATING AN 'hrct' RESOURCE USING RESORCERER


STRUCTURE OF A COMPILED 'hrct' RESOURCE
Header component
HELP MANAGER VERSION
OPTIONS
BALLOON DEFINITION FUNCTION
VARIATION CODE
HOT RECTANGLE COMPONENT COUNT
FIRST HOT RECTANGLE COMPONENT
LAST HOT RECTANGLE COMPONENT
Help Manager version
A number of options. 0, below, is irrelevant. 1 is not relevant for static windows. 2 and 3 relate to the three different ways that the Help Manager draws and removes balloons. 4 is used in 'hwin' resources only.
Resource ID of the window definition function (WDEF) used for drawing help balloons. The standard WDEF's resource ID is 126. This can be specified by 0 in Resorcerer.
Variation code for WDEF. Governs the location of the balloon's tip.
The number of hot rectangle components defined in the rest of this resource.
STRUCTURE OF HOT RECTANGLE COMPONENT
SIZE
TYPE OF DATA
TIPS COORDINATES
HOT RECTANGLE
TEXT STRING
ALIGNMENT BYTES
Pascal string in this component
Coordinates of balloon's tip
Coordinates of hot rectangle
Help message
The structure of the hot rectangle component depends on the item chosen in the Message Type pop-up menu in the Resorcerer editing window below, which sets the TYPE OF DATA field. The pop-up menu items specify the format of the help balloon messages. The available formats are as follows:
Use these strings — Use the string specified within this component of this 'hrct' resource. (Specified in this example.)
Use 'PICT' resources — Use the picture stored in the specified 'PICT' resource.
Use 'STR#' resources — Use the specified text string stored in the specified 'STR#' resource.
Used styled text resources — Use the styled text stored in the specified 'TEXT' and 'styl' resources.
Use 'STR ' resources — Use the text string stored in the specified 'STR ' resource.
Skip missing item — No help message. Skip this item.
RESORCERER 'hrct' RESOURCE EDITING WINDOW
New hrct 128 from MyApp.rsrc
Balloon help version Latest=2
5-31. Reserved 0
4. For 'hwin's, match string anywhere in title Off
3. Create window, restore bits, and cause update Off
2. Don't create window, restore bits, no update Off
1. Pretend window port origin is set to (0,0) Off
0. Treat resource IDs as sub-IDs for owned resources of a D/
Balloon 'WDEF' Resource ID Standard balloons=0
Balloon variation code (tip position) Along left side at top=0
Hot rectangles 2
Hot rectangles #1
Number of bytes to next record
Message type Use this string=1
Tip (x,y)=(0,0)
Hot rect (t,l,b,r)=(10,10,50,50)
Enabled message "This is the hot rectangle for the 'hrct' resource de
New
Edit
Cancel

The `'hwin'` (window help) resource is used to associate the help balloons defined in an `'hrct'` resource with a particular window. All `'hrct'` resources must have resource IDs equal to or greater than 128. Fig 14 shows an `'hwin'` resource being created using Resorcerer.

Figure 14. CREATING AN 'hwin' RESOURCE USING RESORCERER


STRUCTURE OF A COMPILED 'hwin' RESOURCE
HELP MANAGER VERSION
OPTIONS
WINDOW COMPONENT COUNT
RESOURCE ID OF 'hrct' OR 'hdlg' RESOURCE
TYPE OF ASSOCIATED RESOURCE ('hrct' OR 'hdlg')
LENGTH OF COMPARISON STRING, OR A windowKind VALUE
WINDOW TITLE STRING FOR COMPARISON
ALIGNMENT BYTES
RESOURCE ID OF 'hrct' OR 'hdlg' RESOURCE
TYPE OF ASSOCIATED RESOURCE ('hrct' OR 'hdlg')
LENGTH OF COMPARISON STRING, OR A windowKind VALUE
WINDOW TITLE STRING FOR COMPARISON
ALIGNMENT BYTES
Header component
Help Manager version
One option only. (0, 1, 2, and 3, below, are irrelevant to 'hwin' resources.) 4 must be set to On to match windows containing a specified number of sequential characters starting with any character in the window title. If 4 is Off, the Help Manager matches characters starting with the first character of the window title.
The number of window components defined in the rest of the resource.
Specifies the resource ID of the associated 'hrct' resource that specifies the help messages for the window.
Specifies the type of the associated resource ('hrct' for static windows).
Specifies the length of the comparison string or a windowKind value. If the integer of this element is positive, this is the number of characters used for matching this component to a window's title. If the integer is negative, this is the value used for matching this component to a window by the value in the windowKind field of the window's colour window structure. (See below.)
Specifies the window title string. If the previous field is a positive integer, this field consists of characters that the Help Manager uses to match this component to a window by the window's title. If the previous field is negative, this is an empty string.
First window item component
Last window item component
The 'hwin' resource identifies windows by their titles or by the value in the windowKind field of the window's colour window structure. Accordingly, your window's colour window structure must specify either a title or a windowKind value that adequately distinguishes it from other windows. (Note that windowKind values of 0, 1, and 3 to 7 are reserved by the system and that dialog and alert boxes have 2 assigned to their windowKind field. Your ability to distinguish between untitled dialog and alert boxes is thus somewhat limited.)
You can list all of your windows within one 'hwin' resource, or you can create separate 'hwin' resources for your separate windows. You need separate 'hwin' resources for windows that require different options (for example, one requires a precise string match, another matches the string anywhere in the title.)
RESORCERER 'hwin' RESOURCE EDITING WINDOW
New hwin 128 from MyApp.rsrc
Balloon help version Latest=2
5-31. Reserved 0
4. For 'hwin's, match string anywhere in title Off
3. Create window, restore bits, and cause update Off
2. Don't create window, restore bits, no update Off
1. Pretend window port origin is set to (0,0) Off
0. Treat resource IDs as sub-IDs for owned resources of a D/
Static windows 2
Static windows #1
Resource ID of following help type 128
Help resource type hrct='hrct'
Length of comparison string (if <0 ==> -windowKind) 5
Window title comparison string "My App"
Static windows #2
Resource ID of following help type 129
Help resource type hrct='hrct'
Length of comparison string (if <0 ==> -windowKind) 4
Window title comparison string "Find"
New
Edit
Cancel

Main Window Manager Constants, Data Types, and Functions

Constants

Window Types

```
kWindowDocumentProc               = 1024  Document windows
kWindowGrowDocumentProc           = 1025
kWindowVertZoomDocumentProc       = 1026
kWindowVertZoomGrowDocumentProc   = 1027
kWindowHorizZoomDocumentProc      = 1028
kWindowHorizZoomGrowDocumentProc  = 1029
kWindowFullZoomDocumentProc       = 1030
kWindowFullZoomGrowDocumentProc   = 1031
kWindowPlainDialogProc            = 1040  Dialogs and Alerts
kWindowShadowDialogProc           = 1041
kWindowModalDialogProc            = 1042
kWindowMovableModalDialogProc     = 1043
kWindowAlertProc                  = 1044
kWindowMovableAlertProc           = 1045
kWindowMovableModalGrowProc       = 1046
kWindowFloatProc                  = 1057  Floating windows
kWindowFloatGrowProc              = 1059
kWindowFloatVertZoomProc          = 1061
kWindowFloatVertZoomGrowProc      = 1063
kWindowFloatHorizZoomProc         = 1065
kWindowFloatHorizZoomGrowProc     = 1067
kWindowFloatFullZoomProc          = 1069
kWindowFloatFullZoomGrowProc      = 1071
kWindowFloatSideProc              = 1073
kWindowFloatSideGrowProc          = 1075
kWindowFloatSideVertZoomProc      = 1077
kWindowFloatSideVertZoomGrowProc  = 1079
kWindowFloatSideHorizZoomProc     = 1081
kWindowFloatSideHorizZoomGrowProc = 1083
kWindowFloatSideFullZoomProc      = 1085
kWindowFloatSideFullZoomGrowProc  = 1087
```

Window Kind

```
kDialogWindowKind      = 2
kApplicationWindowKind = 8
```

Window Part Codes Returned by FindWindow

```
inDesk                = 0
inNoWindow            = 0
inMenuBar             = 1
inSysWindow           = 2
inContent             = 3
inDrag                = 4
inGrow                = 5
inGoAway              = 6
inZoomIn              = 7
inZoomOut             = 8
inCollapseBox         = 11
inProxyIcon           = 12
```

Regions Codes Passed to GetWindowRegion

```
kWindowTitleBarRgn     = 0
kWindowTitleTextRgn    = 1
kWindowCloseBoxRgn     = 2
kWindowZoomBoxRgn      = 3
kWindowDragRgn         = 5
kWindowGrowRgn         = 6
kWindowCollapseBoxRgn  = 7
kWindowStructureRgn    = 32
kWindowContentRgn      = 33
kWindowUpdateRgn       = 34
kWindowGlobalPortRgn   = 40
```

Options For ScrollWindowRect and ScrollWindowRegion

```
kScrollWindowNoOptions              = 0
kScrollWindowInvalidate             = (1L << 0)  Add exposed area to window's update region.
KScrollWindowEraseToPortBackground  = (1L << 1)  Erase exposed area using background color/
                                                 pattern of the window's grafport.
```

Data Types

```
typedef struct OpaqueWindowPtr*  WindowPtr;
typedef WindowPtr  WindowRef;
typedef SInt16     WindowPartCode;
typedef UInt16     WindowRegionCode;
typedef UInt32     OptionBits;
typedef OptionBits ScrollWindowOptions;
```

Functions

Creating Windows

```
WindowRef  GetNewCWindow(short windowID,void *wStorage,WindowRef behind);
WindowRef  NewCWindow(void *wStorage,const Rect *boundsRect,ConstStr255Param title,Boolean
           visible,short procID,WindowRef behind,Boolean goAwayFlag,long refCon);
```

Naming Windows

```
void       GetWTitle(WindowRef theWindow,Str255 title);
void       SetWTitle(WindowRef theWindow,ConstStr255Param title);
OSStatus   CopyWindowTitleAsCFString(WindowRef inWindow,CFStringRef *outString);
OSStatus   SetWindowTitleWithCFString(WindowRef inWindow,CFStringRef inString);
```

Displaying Windows

```
Boolean    IsWindowVisible(WindowRef window);
void       ShowWindow(WindowRef theWindow);
void       HideWindow(WindowRef theWindow);
void       ShowHide(WindowRef theWindow,Boolean showFlag);
Boolean    IsWindowHilited(WindowRef window);
void       HiliteWindow(WindowRef theWindow,Boolean fHilite);
void       SelectWindow(WindowRef theWindow);
void       BringToFront(WindowRef theWindow);
void       SendBehind(WindowRef theWindow,WindowRef behindWindow);
void       DrawGrowIcon(WindowRef theWindow);
```

Moving Windows

```
void       MoveWindow(WindowRef theWindow,short hGlobal,short vGlobal,Boolean front);
void       DragWindow(WindowRef theWindow,Point startPt,const Rect *boundsRect);
```

Resizing Windows

```
long        GrowWindow(WindowRef theWindow,Point startPt,const Rect *bBox);
void        SizeWindow(WindowRef theWindow,short w,short h,Boolean fUpdate);
Boolean     ResizeWindow(WindowRef window,Point startPoint,const Rect *sizeConstraints,
            Rect *newContentRect);
```

Zooming Windows

```
Boolean     TrackBox(WindowRef theWindow,Point thePt,short partCode);
void        ZoomWindow(WindowRef theWindow,short partCode,Boolean front);
OSStatus    ZoomWindowIdeal(WindowRef window,SInt16 partCode,Point *ioIdealSize);
Boolean     IsWindowInStandardState(WindowRef window,Point *idealSize,Rect
*idealStandardState);
OSStatus    SetWindowIdealUserState(WindowRef window,Rect *userState);
OSStatus    GetWindowIdealUserState(WindowRef window,Rect *userState);
```

Deallocating Windows

```
Boolean     TrackGoAway(WindowRef theWindow,Point thePt);
void        DisposeWindow(WindowRef theWindow);
```

Collapsing Windows

```
Boolean     IsWindowCollapsable(WindowRef inWindow);
Boolean     IsWindowCollapsed(WindowRef inWindow);
OSStatus    CollapseWindow(WindowRef inWindow,Boolean inCollapseIt);
OSStatus    CollapseAllWindows(Boolean inCollapseEm);
```

Window Kind

```
short       GetWindowKind(WindowRef window);
void        SetWindowKind(WindowRef window,short kind);
```

Window Regions

```
OSStatus    GetWindowRegion(WindowRef inWindow,WindowRegionCode inRegionCode,
            RgnHandle ioWinRgn)
```

Window User State and Standard State

```
Rect*       GetWindowStandardState(WindowRef window,Rect *rect);
Rect*       GetWindowUserState(WindowRef window,Rect *rect);
void        SetWindowStandardState(WindowRef window,const Rect *rect);
void        SetWindowUserState(WindowRef window,const Rect *rect);
Boolean     IsWindowInStandardState(WindowRef window,Point *idealSize,Rect
idealStandardState);
```

Getting Available Window Positioning Bounds

```
OSStatus  GetAvailableWindowPositioningBounds(GDHandle inDevice,Rect *availableRect);
```

Maintaining the Update Region

```
void        BeginUpdate(WindowRef theWindow);
void        EndUpdate(WindowRef theWindow);
Boolean     IsWindowUpdatePending(WindowRef window);
OSStatus    InvalWindowRgn(WindowRef window,RgnHandle region);
OSStatus    InvalWindowRect(WindowRef window, const Rect *bounds);
OSStatus    ValidWindowRgn(WindowRef window,RgnHandle region);
OSStatus    ValidWindowRect(WindowRef window, const Rect *bounds);
```

Retrieving Mouse Information

```
short      FindWindow(Point thePoint,WindowRef *theWindow);
WindowRef  FrontWindow(void);
```

Window Reference Constant, Variant, and Picture

```
long       GetWRefCon(WindowRef theWindow);
void       SetWRefCon(WindowRef theWindow,long data);
short      GetWVariant(WindowRef theWindow);
void       SetWindowPic(WindowRef theWindow,PicHandle pic);
PicHandle  GetWindowPic(WindowRef theWindow);
```

Window List

```
WindowRef  GetWindowList(void);
WindowRef  GetNextWindow(WindowRef window);
```

Window's Graphics Port

```
CGrafPtr   GetWindowPort(WindowRef window);
void       SetPortWindowPort(WindowRef window);
WindowRef  GetWindowFromPort(CGrafPtr port);
Rect*      GetWindowPortBounds(WindowRef window,Rect *bounds);
```

Gray Region

```
RgnHandle  GetGrayRgn(void);
```

Scrolling Pixels in the Window Content Region

```
OSStatus   ScrollWindowRect(WindowRef   window,const Rect *scrollRect,SInt16 hPixels,
           SInt16 vPixels,ScrollWindowOptions options,RgnHandle outExposedRgn);
OSStatus   ScrollWindowRegion(WindowRef window,RgnHandle scrollRgn,
           SInt16 hPixels,vPixels,ScrollWindowOptions options,RgnHandle outExposedRgn);
```

Demonstration Program Windows1 Listing

```
// ****************************************************************************************
// Windows1.c                                                          CLASSIC EVENT MODEL
// ****************************************************************************************
//
// This program:
//
// •  Allows the user to open any number of kWindowFullZoomGrowDocumentProc windows, up to the
//    maximum specified by the constant assigned to the symbolic name kMaxWindows, using the
//    File menu Open Command or its keyboard equivalent.
//
// •  Allows the user to close opened windows using the close box/button, the File menu Close
//    command or the Close command's keyboard equivalent.
//
// •  Adds menu items representing each window to a Windows menu as each window is opened.
//    (A keyboard equivalent is included in each menu item for windows 1 to 9.)
//
// •  Deletes menu items from the Windows menu as each window is closed.
//
// •  Fills each window with a plain color pattern as a means of proving, for demonstration
//    purposes, the window update process.
//
// •  Facilitates activation of a window by mouse selection.
//
// •  Facilitates activation of a window by Windows menu selection.
//
// •  Correctly performs all dragging, zooming and sizing operations.
//
// •  On Mac OS 8/9, demonstrates the provision of balloon help for static windows.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  An 'MBAR' resource, and 'MENU' resources for Apple/Application, File, Edit and Windows
//    menus, (preload non-purgeable).
//
// •  A 'WIND' resource (purgeable) (initially not visible).
//
// •  A 'STR#' resource containing error strings and the window title (purgeable).
//
// •  An 'hrct' resource and an 'hwin' resource for balloon help (both purgeable).
//
// •  Ten 'ppat' (pixel pattern) resources (purgeable), which are used to draw a plain color
//    pattern in the windows.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// ****************************************************************************************

// ..........................................................................................................................................................................
includes

#include <Carbon.h>

// ..........................................................................................................................................................................
defines

#define rMenubar          128
#define mAppleApplication 128
#define  iAbout           1
#define mFile             129
#define  iNew             1
#define  iClose           4
#define  iQuit            12
#define mWindows          131
#define rNewWindow        128
#define rStringList       128
#define  sUntitled        1
#define  eMaxWindows      2
#define  eFailWindow      4
#define  eFailMenus       5
#define  eFailMemory      6
#define rPixelPattern     128
#define kMaxWindows       10
#define kScrollBarWidth   15
#define MAX_UINT32        0xFFFFFFFF
#define MIN(a,b)          ((a) < (b) ? (a) : (b))
#define topLeft(r)        (((Point *) &(r))[0])

// ..................................................................................................................................................... global variables

Boolean   gRunningOnX = false;
```

```
Boolean   gDone;
SInt32    gUntitledWindowNumber = 0;
SInt32    gCurrentNumberOfWindows = 0;
WindowRef gWindowRefArray[kMaxWindows + 2];

// ..................................................................................................................................................................... function prototypes

void    main                    (void);
void    doPreliminaries         (void);
OSErr   quitAppEventHandler     (AppleEvent *,AppleEvent *,SInt32);
void    eventLoop               (void);
void    doEvents                (EventRecord *);
void    doMouseDown             (EventRecord *);
void    doUpdate                (EventRecord *);
void    doUpdateWindow          (EventRecord *);
void    doActivate              (EventRecord *);
void    doActivateWindow        (WindowRef,Boolean);
void    doOSEvent               (EventRecord *);
void    doMenuChoice            (SInt32);
void    doFileMenu              (MenuItemIndex);
void    doWindowsMenu           (MenuItemIndex);
void    doNewWindow             (void);
void    doCloseWindow           (void);
void    doInvalidateScrollBarArea (WindowRef);
void    doConcatPStrings        (Str255,Str255);
void    doErrorAlert            (SInt16);
Boolean eventFilter             (DialogPtr,EventRecord *,SInt16 *);

// ********************************************************************************* main

void  main(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  SInt16        a;
  // ..................................................................................................................................................... do preliminaries

  doPreliminaries();

  // ..................................................................................................................................... set up menu bar and menus

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    doErrorAlert(eFailMenus);
  SetMenuBar(menubarHdl);
  DrawMenuBar();

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
    }

    gRunningOnX = true;
  }

  // ..........................................................................................................................................................
initialize window reference array

  for(a=0;a<kMaxWindows+2;a++)
    gWindowRefArray[a] = NULL;

  // ..................................................................................................................................... enter eventLoop

  eventLoop();
}

// ****************************************************************************** doPreliminaries

void  doPreliminaries(void)
{
  OSErr osError;

  MoreMasterPointers(224);
  InitCursor();
  FlushEvents(everyEvent,0);

  osError = AEInstallEventHandler(kCoreEventClass,kAEQuitApplication,
                          NewAEEventHandlerUPP((AEEventHandlerProcPtr) quitAppEventHandler),
                          0L,false);
  if(osError != noErr)
    ExitToShell();
}
// ********************************************************************************* doQuitAppEvent
```

```
OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  OSErr    osError;
  DescType returnedType;
  Size     actualSize;

  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,NULL,0,
                              &actualSize);

  if(osError == errAEDescNotFound)
  {
    gDone = true;
    osError = noErr;
  }
  else if(osError == noErr)
    osError = errAEParamMissed;

  return osError;
}

// ********************************************************************************** eventLoop

void  eventLoop(void)
{
  EventRecord eventStructure;

  gDone = false;

  while(!gDone)
  {
    if(WaitNextEvent(everyEvent,&eventStructure,MAX_UINT32,NULL))
      doEvents(&eventStructure);
  }
}

// ********************************************************************************** doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
  switch(eventStrucPtr->what)
  {
    case kHighLevelEvent:
      AEProcessAppleEvent(eventStrucPtr);
      break;

    case mouseDown:
      doMouseDown(eventStrucPtr);
      break;

    case keyDown:
      if((eventStrucPtr->modifiers & cmdKey) != 0)
        doMenuChoice(MenuEvent(eventStrucPtr));
      break;
    case updateEvt:
      doUpdate(eventStrucPtr);
      break;

    case activateEvt:
      doActivate(eventStrucPtr);
      break;

    case osEvt:
      doOSEvent(eventStrucPtr);
      break;
  }
}

// ********************************************************************************** doMouseDown

void  doMouseDown(EventRecord *eventStrucPtr)
{
  WindowRef      windowRef;
  WindowPartCode partCode, zoomPart;
  BitMap         screenBits;
  Rect           constraintRect, mainScreenRect;
  Point          standardStateHeightAndWidth;

  partCode = FindWindow(eventStrucPtr->where,&windowRef);

  switch(partCode)
  {
    case inMenuBar:
      doMenuChoice(MenuSelect(eventStrucPtr->where));
      break;

    case inContent:
      if(windowRef != FrontWindow())
        SelectWindow(windowRef);
```

```
      break;

    case inDrag:
      DragWindow(windowRef,eventStrucPtr->where,NULL);
      break;

    case inGoAway:
      if(TrackGoAway(windowRef,eventStrucPtr->where) == true)
        doCloseWindow();
      break;

    case inGrow:
      constraintRect.top   = 75;
      constraintRect.left = 205;
      constraintRect.bottom = constraintRect.right = 32767;
      doInvalidateScrollBarArea(windowRef);
      ResizeWindow(windowRef,eventStrucPtr->where,&constraintRect,NULL);
      doInvalidateScrollBarArea(windowRef);
      break;
    case inZoomIn:
    case inZoomOut:
      mainScreenRect = GetQDGlobalsScreenBits(&screenBits)->bounds;
      standardStateHeightAndWidth.v = mainScreenRect.bottom;
      standardStateHeightAndWidth.h = 460;

      if(IsWindowInStandardState(windowRef,&standardStateHeightAndWidth,NULL))
        zoomPart = inZoomIn;
      else
        zoomPart = inZoomOut;

      if(TrackBox(windowRef,eventStrucPtr->where,partCode))
        ZoomWindowIdeal(windowRef,zoomPart,&standardStateHeightAndWidth);
      break;
  }
}

// ********************************************************************************** doUpdate

void  doUpdate(EventRecord *eventStrucPtr)
{
  WindowRef windowRef;

  windowRef = (WindowRef) eventStrucPtr->message;

  BeginUpdate(windowRef);

  SetPortWindowPort(windowRef);
  doUpdateWindow(eventStrucPtr);

  EndUpdate(windowRef);
}

// ************************************************************************** doUpdateWindow

void  doUpdateWindow(EventRecord *eventStrucPtr)
{
  WindowRef    windowRef;
  RgnHandle    visibleRegionHdl;
  Rect         theRect;
  SInt32       windowRefCon;
  PixPatHandle pixpatHdl;
  RGBColor     whiteColor = {  0xFFFF,0xFFFF,0xFFFF } ;
  SInt16       a;

  windowRef = (WindowRef) eventStrucPtr->message;

  visibleRegionHdl = NewRgn();
  GetPortVisibleRegion(GetWindowPort(windowRef),visibleRegionHdl);
  EraseRgn(visibleRegionHdl);
  DisposeRgn(visibleRegionHdl);

  GetWindowPortBounds(windowRef,&theRect);
  theRect.right  -= kScrollBarWidth;
  theRect.bottom -= kScrollBarWidth;
  windowRefCon = GetWRefCon(windowRef);
  pixpatHdl = GetPixPat(rPixelPattern + windowRefCon);
  FillCRect(&theRect,pixpatHdl);
  DisposePixPat(pixpatHdl);

  DrawGrowIcon(windowRef);

  RGBForeColor(&whiteColor);
  TextSize(10);

  if(!gRunningOnX)
  {
    for(a=0;a<2;a++)
    {
      SetRect(&theRect,a*90+10,10,a*90+90,33);
```

```
      FrameRect(&theRect);
      MoveTo(a*90+18,25);

      DrawString("\pHot Rectangle");
    }
  }
}

// ********************************************************************** doActivate

void  doActivate(EventRecord *eventStrucPtr)
{
  WindowRef windowRef;
  Boolean   becomingActive;

  windowRef = (WindowRef) eventStrucPtr->message;

  becomingActive = ((eventStrucPtr->modifiers & activeFlag) == activeFlag);

  doActivateWindow(windowRef,becomingActive);
}

// ********************************************************************** doActivateWindow

void  doActivateWindow(WindowRef windowRef,Boolean becomingActive)
{
  MenuRef windowsMenu;
  SInt16  menuItem, a = 1;

  windowsMenu = GetMenuRef(mWindows);

  while(gWindowRefArray[a] != windowRef)
    a++;
  menuItem = a;

  if(becomingActive)
    CheckMenuItem(windowsMenu,menuItem,true);
  else
    CheckMenuItem(windowsMenu,menuItem,false);
  DrawGrowIcon(windowRef);
}

// ********************************************************************** doOSEvent

void  doOSEvent(EventRecord *eventStrucPtr)
{
  switch((eventStrucPtr->message >> 24) & 0x000000FF)
  {
    case suspendResumeMessage:
      if((eventStrucPtr->message & resumeFlag) == 1)
        SetThemeCursor(kThemeArrowCursor);
      break;
  }
}

// ********************************************************************** doMenuChoice

void  doMenuChoice(SInt32 menuChoice)
{
  MenuID        menuID;
  MenuItemIndex menuItem;

  menuID = HiWord(menuChoice);
  menuItem = LoWord(menuChoice);

  if(menuID == 0)
    return;

  switch(menuID)
  {
    case mAppleApplication:
      if(menuItem == iAbout)
        SysBeep(10);
      break;

    case mFile:
      doFileMenu(menuItem);
      break;

    case mWindows:
      doWindowsMenu(menuItem);
      break;
  }

  HiliteMenu(0);
}

// ********************************************************************** doFileMenu
```

```
void  doFileMenu(MenuItemIndex menuItem)
{
  switch(menuItem)
  {
    case iNew:
      doNewWindow();
      break;

    case iClose:
      doCloseWindow();
      break;

    case iQuit:
      gDone = true;
      break;
  }
}

// ********************************************************************************* doWindowsMenu

void  doWindowsMenu(MenuItemIndex menuItem)
{
  WindowRef windowRef;

  windowRef = gWindowRefArray[menuItem];
  SelectWindow(windowRef);
}

// ********************************************************************************** doNewWindow

void  doNewWindow(void)
{
  WindowRef windowRef;
  Str255    untitledString;
  Str255    numberAsString = "\p1";
  Rect      availableBoundsRect, portRect;
  SInt16    windowHeight;
  MenuRef   windowsMenu;

  if(gCurrentNumberOfWindows == kMaxWindows)
  {
    doErrorAlert(eMaxWindows);
    return;
  }

  if(!(windowRef = GetNewCWindow(rNewWindow,NULL,(WindowRef) -1)))
    doErrorAlert(eFailWindow);

  GetIndString(untitledString,rStringList,sUntitled);
  gUntitledWindowNumber += 1;
  if(gUntitledWindowNumber > 1)
  {
    NumToString(gUntitledWindowNumber,numberAsString);
    doConcatPStrings(untitledString,numberAsString);
  }

  SetWTitle(windowRef,untitledString);
  GetAvailableWindowPositioningBounds(GetMainDevice(),&availableBoundsRect);
  GetWindowPortBounds(windowRef,&portRect);
  SetPortWindowPort(windowRef);
  LocalToGlobal(&topLeft(portRect));
  windowHeight = (availableBoundsRect.bottom - portRect.top) - 3;
  if(!gRunningOnX)
    windowHeight -= 27;
  SizeWindow(windowRef,460,windowHeight,false);

  ShowWindow(windowRef);

  if(gUntitledWindowNumber < 10)
  {
    doConcatPStrings(untitledString,"\p/");
    doConcatPStrings(untitledString,numberAsString);
  }
  windowsMenu = GetMenuRef(mWindows);
  InsertMenuItem(windowsMenu,untitledString,CountMenuItems(windowsMenu));

  SetWRefCon(windowRef,gCurrentNumberOfWindows);

  gCurrentNumberOfWindows ++;
  gWindowRefArray[gCurrentNumberOfWindows] = windowRef;

  if(gCurrentNumberOfWindows == 1)
  {
    EnableMenuItem(GetMenuRef(mFile),iClose);
    EnableMenuItem(GetMenuRef(mWindows),0);
    DrawMenuBar();
  }
}
```

```
// ********************************************************************************* doCloseWindow

void  doCloseWindow(void)
{
  WindowRef windowRef;
  MenuRef   windowsMenu;
  SInt16    a = 1;

  windowRef = FrontWindow();
  DisposeWindow(windowRef);
  gCurrentNumberOfWindows --;

  windowsMenu = GetMenuRef(mWindows);
  while(gWindowRefArray[a] != windowRef)
    a++;
  gWindowRefArray[a] = NULL;
  DeleteMenuItem(windowsMenu,a);

  for(a=1;a<kMaxWindows+1;a++)
  {
    if(gWindowRefArray[a] == NULL)
    {
      gWindowRefArray[a] = gWindowRefArray[a+1];
      gWindowRefArray[a+1] = NULL;
    }
  }

  if(gCurrentNumberOfWindows == 0)
  {
    DisableMenuItem(GetMenuRef(mFile),iClose);
    DisableMenuItem(GetMenuRef(mWindows),0);
    DrawMenuBar();
  }
}

// ************************************************************** doInvalidateScrollBarArea

void  doInvalidateScrollBarArea(WindowRef windowRef)
{
  Rect tempRect;

  SetPortWindowPort(windowRef);

  GetWindowPortBounds(windowRef,&tempRect);
  tempRect.left = tempRect.right - kScrollBarWidth;
  InvalWindowRect(windowRef,&tempRect);

  GetWindowPortBounds(windowRef,&tempRect);
  tempRect.top = tempRect.bottom - kScrollBarWidth;
  InvalWindowRect(windowRef,&tempRect);
}

// ********************************************************************************* doConcatPStrings

void  doConcatPStrings(Str255 targetString,Str255 appendString)
{
  SInt16 appendLength;

  appendLength = MIN(appendString[0],255 - targetString[0]);

  if(appendLength > 0)
  {
    BlockMoveData(appendString+1,targetString+targetString[0]+1,(SInt32) appendLength);
    targetString[0] += appendLength;
  }
}

// ********************************************************************************* doErrorAlert

void  doErrorAlert(SInt16 errorType)
{
  AlertStdAlertParamRec paramRec;
  ModalFilterUPP        eventFilterUPP;
  Str255                labelText;
  Str255                narrativeText;
  SInt16                itemHit;
  eventFilterUPP = NewModalFilterUPP((ModalFilterProcPtr) eventFilter);

  paramRec.movable       = true;
  paramRec.helpButton    = false;
  paramRec.filterProc    = eventFilterUPP;
  paramRec.defaultText   = (StringPtr) kAlertDefaultOKText;
  paramRec.cancelText    = NULL;
  paramRec.otherText     = NULL;
  paramRec.defaultButton = kAlertStdAlertOKButton;
  paramRec.cancelButton  = 0;
  paramRec.position      = kWindowAlertPositionMainScreen;

  GetIndString(labelText,rStringList,errorType);
```

```
  if(errorType == eMaxWindows)
  {
    GetIndString(narrativeText,rStringList,errorType + 1);
    StandardAlert(kAlertCautionAlert,labelText,narrativeText,&paramRec,&itemHit);
    DisposeModalFilterUPP(eventFilterUPP);
  }
  else
  {
    StandardAlert(kAlertStopAlert,labelText,0,&paramRec,&itemHit);
    ExitToShell();
  }
}

// ********************************************************************** eventFilter

Boolean  eventFilter(DialogPtr dialogPtr,EventRecord *eventStrucPtr,SInt16 *itemHit)
{
  Boolean handledEvent = false;

  if((eventStrucPtr->what == updateEvt) &&
     ((WindowRef) eventStrucPtr->message != GetDialogWindow(dialogPtr)))
  {
    doUpdate(eventStrucPtr);
  }

  return handledEvent;
}

// ********************************************************************************
```

Demonstration Program Windows1 Comments

When this program is run, the user should

- Open and close windows using both the Open and Close commands from the File menu and their keyboard equivalents, noting that whenever a window is opened or closed, a menu item representing that window is added to or deleted from the Windows menu.
- Note that keyboard equivalents are added to the menu items in the Windows menu for the first nine windows opened.
- Activate individual windows by both clicking the content region and pressing the keyboard equivalent for the window.
- Send the application to the background and bring it to the foreground, noting window activation/deactivation.
- Zoom, close, and resize windows using the zoom box/button, close box/button and size box/resize control, noting window updating and activation.
- On Mac OS X, note that when a window is zoomed to the standard state, the zoom is constrained by the current height of the Dock.
- On Mac OS 8/9, choose Show Balloons from the Help menu and move the cursor over the hot rectangles in the frontmost window.

If an attempt is made to open more than 10 windows, a movable modal alert appears.

defines

The first nine `#defines` establish constants representing menu IDs and resources, and window and menu bar resources. The next six establish constants representing the resource ID of a `'STR#'` resource and the various strings in that resource. `rPixelPattern` represents the resource ID of the first of ten `'ppat'` (pixel pattern) resources. `kMaxWindows` controls the maximum number of windows allowed to be open at one time.

`MAX_UINT32` is defined as the maximum possible `UInt32` value. (This will be assigned to `WaitNextEvent`'s sleep parameter.) The two fairly common macros which follow are required by, respectively, the string concatenation function `doConcatPStrings` and the

window creation function `doNewWindow`.

Global Variables

The global variable `gRunningOnX` will be set to true if the program is running on Mac OS X. `gDone`, when set to true, causes the main event loop to be exited and the program to terminate.

`gUntitledWindowNumber` keeps track of the window numbers to be inserted into the window's title bar. This number is incremented each time a new window is opened. `gCurrentNumberOfWindows` keeps track of how many windows are open at any one time.

In this program, `CreateStandardWindowMenu` is not used to create the Window menu. The Window menu is created in the same way as the other menus, and managed by the program. `gWindowRefArray[]` is central to the matter of maintaining an association between item numbers in the Window menu and the windows to which they refer, regardless of how many windows are opened and closed, and in what sequence. When, for example, a Window menu item is chosen, the program must be able to locate the window object for the window represented by that menu item number so as to activate the correct window.

The strategy adopted by this program is to assign the references for each opened window to the elements of `gWindowRefArray[]`, starting with `gWindowRefArray[1]` and leaving `gWindowRefArray[0]` unused. If, for example, six windows are opened in sequence, `gWindowRefArray[1]` to `gWindowRefArray[6]` are assigned the references to the window objects for each of those six windows. (At the same time, menu items representing each of those windows are progressively added to the Window menu.)

If, say, the third window opened is then closed, `gWindowRefArray[3]` is set to NULL and the window object references in `gWindowRefArray[4]` to `gWindowRefArray[6]` are moved down in the array to occupy `gWindowRefArray[3]` to `gWindowRefArray[5]`. Since the Window menu item for the third window is deleted from the menu when the window is closed, there remains five windows and their associated menu items, the "compaction" of the array having maintained a direct relationship between the number of the array element to which each window reference is assigned and the number of the menu item for that window.

main

The first action is to call `doPreliminaries`, which performs certain preliminary actions common to most applications.

The next block sets up the menus. Note that error handling involving the invocation of

alerts is introduced in this program. If an error occurs, the function `doErrorAlert` is called to display either a stop or caution movable modal alert advising of the nature of the error.

In the final three lines, `gWindowRefArray[]` is initialized and the main event loop is entered.

doPreliminaries

Note that `MoreMasterPointers` is called with 224 passed in the `inCount` parameter to provide sufficient master pointers for this program.

eventLoop

`eventLoop` will exit when `gDone` is set to true, which occurs when the user selects Quit from the File menu. (As an aside, note that the sleep parameter in the `WaitNextEvent` call is set to `MAX_UINT32`, which is defined as the maximum possible `UInt32` value.)

doEvents

`doEvents` switches according to the event type received.

`mouseDown`, `updateEvt`, `activateEvt` and `osEvt` events are of significance to the windows aspects of this demonstration. `keyDown` events are significant only with regard to File and Window menu keyboard equivalents.

doMouseDown

`doMouseDown` continues the processing of `mouseDown` events, switching according to the part code.

The `inContent` case results in a call to `SelectWindow` if the window in which the mouse-down occurred is not the front window. `SelectWindow`

- Unhighlights the currently active window, brings the specified window to the front and highlights it.
- Generates activate events for the two windows.
- Moves the previously active window to a position immediately behind the specified window.

The `inDrag` case results in a call to `DragWindow`, which retains control until the user

releases the mouse button. The third parameter in the `DragWindow` call establishes the limits, in global screen coordinates, within which the user is allowed to drag the window. In Carbon, NULL may be passed in this parameter. This has the effect of setting the third parameter to the bounding rectangle of the desktop region (also known as the "gray region").

The `inGoAway` case results in a call to `TrackGoAway`, which retains control until the user releases the mouse button. If the pointer is still within the close box/button when the mouse button is released, the function `doCloseWindow` is called.

At the `inGrow` case, the first three lines establish the resizing constraints. The top and left fields of the `Rect` variable `constraintRect` are assigned values representing the minimum height and width to which the window may be resized. The bottom and right fields, which establish the maximum height and width, are assigned the maximum possible `SInt16` value. (Since the mouse cursor cannot be moved beyond the edges of the screen (or screens in a multimonitor system), these values mean that the window can be resized larger to the limits of mouse cursor movement.)

`ResizeWindow` retains control until the user releases the mouse button. When the mouse button is released, `ResizeWindow` redraws the window frame (that is, all but the content region) in the new size and, where window height and/or width has been increased, adds the newly exposed areas of the content region to update region (on Mac OS 8/9). (Note that, in Carbon, the fourth (`newContentRect`) parameter may be set to NULL if the new dimension of the window's content region is not required.)

The call to `ResizeWindow` is bracketed by two calls to the function `doInvalidateScrollBarArea`. In this program, scroll bars are not used but it has been decided to, firstly, limit update drawing to the window's content region less the areas normally occupied by scroll bars and, secondly, to use `DrawGrowIcon` to draw the draw scroll bar delimiting lines. (For Mac OS 8/9, this is the usual practice for windows with a size box but no scroll bars. The `DrawGrowIcon` call is ignored on Mac OS X.)

The first call to `doInvalidateScrollBarArea` is necessary to cater for the case where the window is resized larger. If this call is not made, the scroll bar areas prior to the resize will not be redrawn by the window updating function unless these areas are programmatically added to the new update region created by the Window Manager as a result of the resizing action.

The second call to `doInvalidateScrollBarArea` is necessary to cater for the case where the window is resized smaller. This call works in conjunction with the `EraseRgn` call in the function `doUpdateWindow`. The call to `doInvalidateScrollBarArea` results in an update event being generated, and the call to `EraseRgn` in the `doUpdateWindow` function causes the update region (that is, in this case, the scroll bar areas) to be erased. (Remember that, on Mac OS 8/9, between the calls to `BeginUpdate and EndUpdate`, the visible region equates to the update region and that QuickDraw

limits its drawing to the update region.)

At the `inZoomIn`/`inZoomOut` case, the first action is to assign the desired height and width of the windows's standard state content region to the fields of a Point variable. This variable is then passed in the second parameter of a call to `IsWindowInStandardState`, which sets the variable `zoomPart` to either true or false depending on whether the window is currently in the standard state or the user state. `TrackBox` is then called, taking control until the user releases the mouse button. If the mouse button is released while the pointer is still within the zoom box, `ZoomWindowIdeal` is called to zoom the window in accordance with human interface guidelines. The second parameter tells `ZoomWindow` whether to zoom out to the standard state or in to the user state.

doUpdate

On Mac OS 8/9 and Mac OS X, an update event will be received

- When the window is created
- When the window is resized larger
- When the window is resized smaller (because of calls to `InvalWindowRect` in the function `doInvalidateScrollBarArea`)
- When the window is zoomed

On Mac OS 8/9, update events will also be received when a window has a non-empty update region

`doUpdate` attends to basic window updating. On Mac OS 8/9, the call to `BeginUpdate` clips the visible region to the intersection of the visible region and the update region. The visible region is now a sort of proxy for the update region. The graphics port is then set before the function `doUpdateWindow` is called to redraw the content region. On Mac OS 8/9, the `EndUpdate` call restores the window's true visible region. (The calls to `BeginUpdate` and `EndUpdate` are ignored on Mac OS X.)

doUpdateWindow

`doUpdateWindow` is concerned with redrawing the window's contents less the scroll bar areas.

The first action is to retrieve the window object reference from the message field of the event structure.

The next block retrieves the handle to the window's visible region, following which `EraseRgn` is called for reasons explained at `doMouseDown`.

The window's graphics port's bounding rectangle is then retrieved, following which the right and bottom fields are adjusted to exclude the scroll bar areas. The next four lines fill this rectangle with a plain color pattern provided by a `'ppat'` resource, simply as a means of proving the correctness of the window updating process.

Note the call to `GetWRefCon`, which retrieves the window's reference constant stored in the window object. As will be seen, whenever a new window is opened, a value between 1 and `kMaxWindows` is stored as a reference constant in the window object. In this function, this is just a convenient number to be added to the base resource ID (128) in the single parameter of the `GetPixPat` call, ensuring that `FillCRect` has a different pixel pattern to draw in each window.

The call to `DrawGrowIcon` draws the scroll bar delimiting lines (on Mac OS 8/9). Note that this call, the preceding `EraseRgn` call, and the calls to `doInvalidateScrollbarArea` are made for "cosmetic" purposes only and would not be required if the window contained scroll bars.

If the program is running on Mac OS 8/9, the remaining lines draw two rectangles and some text in the windows to visually represent to the user the otherwise invisible "hot rectangles" defined in the `'hrct'` resource and associated with the window by the `'hwin'` resource. When Show Balloons is chosen from the Help menu, the help balloons will be invoked when the cursor moves over these rectangles.

doActivate

`doActivate` attends to those aspects of window activation not handled by the Window Manager.

The modifiers field of the event structure is tested to determine whether the window in question is being activated or deactivated. The result of this test is passed as a parameter in the call to the function `doActivateWindow`.

doActivateWindow

In this demonstration, the remaining actions carried out in response to an `activateEvt` are limited to placing/removing checkmarks in/from items in the Window menu.

The first step is to associate the received `WindowRef` with its item number in the Window menu. At the while loop, the array maintained for that purpose is searched until a match is found. The array element number at which the match is found correlates directly with the menu item number; accordingly, this is assigned to the variable `menuItem`, which is used in the following `CheckMenuItem` calls. Whether the checkmark is added or removed depends on whether the window in question is being activated or deactivated, a condition passed to the call to `doActivateWindow` as its second parameter.

The call to `DrawGrowIcon` ensures that the scroll bar area delimiting lines will be drawn in gray when the window is deactivated (on Mac OS 8/9).

doOSEvent

`doOSEvent` handles operating system events. In this demonstration, action is taken only in the case of resume events. If the event is a resume event, the cursor is set to the arrow shape.

doMenuChoice and doFileMenu

`doMenuChoice` switches according to the menu choices of the user. `doFileMenu` switches according to the File menu item choice of the user.

doWindowsMenu

`doWindowsMenu` takes the item number of the selected Window menu item and, because this equates to the number of the array element in which the associated window object reference is stored, extracts the window object reference associated with the menu item. This is used in the call to `SelectWindow`, which generates the `activateEvts` required to activate and deactivate the appropriate windows.

doNewWindow

`doNewWindow` opens a new window and attends to associated tasks.

In the first block, if the current number of open windows equals the maximum allowable specified by `kMaxWindows`, a caution movable modal alert is called up via the function `doErrorAlert` (with the string represented by `eMaxWindows` displayed) and an immediate return is executed when the user clicks the alert's OK button.

At the next block, the new window is created by the call to `GetNewCWindow`. The third parameter specifies that the window is to be opened in front of all other windows. If the call is not successful for any reason, a stop movable modal alert is called up via the function `doErrorAlert` (with the string represented by `eFailWindow` displayed) and the program terminates when the user clicks the alert's OK button.

The next seven lines create the string which will be used to set the window's title. The code reflects the fact that Aqua Human Interface Guidelines require that a number only be appended to "untitled" for the second and later windows. Accordingly, concatenating a number to the string "untitled" is not affected for the first window created.

`GetIndString` retrieves the string "untitled" from the specified `'STR#'` resource and

the global variable which keeps track of the numbers for the title bar is incremented. If this is not the first window to be created, `NumToString` converts that number to a Pascal string and this string is concatenated to the "untitled" string. The `SetWTitle` call then sets the window's title.

The next block sets adjusts the size of the window before it is displayed. The width is set to 460 pixels and the height is adjusted according to the available screen real estate.

The call to `GetAvailableWindowPositioningBounds` returns, in global coordinates, the available real estate on the main screen (device). This excludes the menu bar and, on Mac OS X, the Dock. The call to `SetPortWindowPort` sets the window's graphics port as the current port, a necessary precursor to the call to `LocalToGlobal`, which converts the top-left (local) coordinates of the port rectangle to global coordinates. The height of the rectangle returned by `GetAvailableWindowPositioningBounds` is then reduced by the distance of the top of the port rectangle from the top of the screen, and then further reduced by three. On Mac OS X, this will cause the bottom of the window to be just above the top of the Dock. If the program is running on Mac OS 8/9, the height is reduced by a further 27 pixels to accommodate the height of the control strip. The call to `SizeWindow` sets the window's size. (The window's location is determined by the positioning specification in the window's `'WIND'` resource.)

The `ShowWindow` call makes the window visible.

The next block adds the metacharacter `\` and the window number to the string used to set the window title (thus setting up the Command key equivalent) before InsertMenuItem is called to create a new menu item to the Window menu. Note that the Command-key equivalent is only added for the first nine windows opened.)

The `SetWRefCon` call stores the value represented by `gCurrentNumberOfWindows` in the window object as the window's reference constant. As previously stated, in this demonstration this is used to select a pixel pattern to draw in each window's content region.

At the next two lines, the variable which keeps track of the current number of opened windows is incremented and the appropriate element of the window reference array is assigned the reference to newly opened window's window object.

The last block enables the Window menu and the Close item in the File menu when the first window is opened.

doCloseWindow

The function `doCloseWindow` closes an open window and attends to associated tasks.

At the first two lines, a reference to the frontmost window's window object is retrieved and passed in the call to DisposeWindow. DisposeWindow removes the window from the screen, removes it from the window list, and discards all its data storage. With the window closed, the global variable that keeps track of the number of windows currently open is decremented.

The next block deletes the associated menu item from the Window menu. At the first four lines, the array element in which the window object reference in question is located is searched out, the element number (which correlates directly with the menu item number) is noted and the element is set to NULL. The call to `DeleteMenuItem` then deletes the menu item.

The for loop "compacts" the array, that is, it moves the contents of all elements above the NULLed element down by one, maintaining the correlation with the Windows menu.

The last block disables the Windows menu and the Close item in the File menu if no windows remain open as a result of this closure.

doInvalidateScrollBarArea

`doInvalidateScrollBarArea` invalidates that part of the window's content area which would ordinarily be occupied by scroll bars. The function simply retrieves the coordinates of the content area into a local `Rect` and reduces this `Rect` to the relevant scroll bar area before invalidating that area, that is, adding it to the window's update region.

doConcatPStrings

The function `doConcatPStrings` concatenates two Pascal strings.

doErrorAlert and eventFilter

`doErrorAlert` displays either a caution alert or a stop alert with a specified string (two strings in the case of the `eMaxWindows` error) extracted from the `'STR#'` resource identified by `rStringList`. `eventFilter` supports `doErrorAlert`.

The creation of alerts using the `StandardAlert` function, and event filter functions, are addressed in Chapter 8.

Chapter 5. Carbon and Universal Procedure Pointers

IN THIS CHAPTER

- Introduction
- The 68LC040 Emulator and the Mixed Mode Manager
- Universal Procedure Pointers and Carbon
- Creation/Disposal Functions Relevant to Demonstration Programs

Introduction

Previous demonstration programs have called a function (`AEInstallEventHandler`) which takes a **universal procedure pointer** as a parameter; however, an explanation of this term has been delayed until this chapter because the `AEInstallEventHandler` calls were only incidental to the main purpose of those demonstrations. The demonstration programs in Chapter 7 are the first in which calls to a system software function which takes a universal procedure pointer as a parameter are central to the demonstration. As a brief but necessary prelude for what is to come, therefore, this chapter addresses the role of universal procedure pointers in the general scheme of things, when and why they were introduced, and their relevance in the Carbon environment.

In a sense, universal procedure pointers are a piece of historical baggage dragged into Carbon by Mac OS 8/9. They were first introduced with the so-called Universal Headers which, in turn, were introduced with the Power Macintosh. They had to with the ability of the Power Macintosh to run applications that use the instruction set of the Motorola 680x0 microprocessor (used in 680x0-based Macintoshes) as well as applications that use the native instruction set of the Power Macintosh's PowerPC microprocessor.

In the Chapter 7 demonstration programs, the system software function that takes a universal procedure pointer is `TrackControl`. This function is called by your application when a mouse-down event occurs in a control, such as a scroll bar. Prior to the introduction of the Power Macintosh and the associated introduction of the Universal Headers, the `TrackControl` prototype looked like this:

```
Sint16  TrackControl(TrackControl(ControlHandle theControl,Point localPoint,
                        ProcPtr actionProc);
```

The `actionProc` a parameter is simply the address of an application-defined function, called a **callback procedure** (or, in C terminology, a **callback function**) that is called repeatedly while the mouse button remains down. In other words, the `TrackControl` function used to take a **procedure pointer** (or, in C terminology, a **function pointer**) in its `actionProc` parameter.

The Universal Headers, which allow you to write Classic API source code capable of being compiled as either 680x0 code or native PowerPC code, changed the prototype for `TrackControl` to

```
ControlPartCode  TrackControl(ControlHandle theControl,Point startPoint,
                              ControlActionUPP actionProc);
```

Notice that the third parameter is now of type `ControlActionUPP`. This means that the `actionProc` parameter now takes a **universal procedure pointer.** This prototype has been carried through to Carbon, hence the necessity to gain a basic understanding of universal procedure pointers.

The 68LC040 Emulator and the Mixed Mode Manager

The Emulator

The ability of the Mac OS 8/9 system software to execute applications that use the instruction set of the Motorola 680x0 microprocessor as well as applications that use the native instruction set of the PowerPC microprocessor is provided by an **emulator** (the 68LC040 Emulator). The emulator, which is essentially a 680x0 microprocessor implemented in software, provides an execution environment that is virtually identical to the execution environment found on 680x0-based Macintoshes.

One important aspect of the 68LC040 emulator is that it made it possible for parts of the system software to remain as 680x0 code while other parts were progressively rewritten, primarily for reasons of speed, as native PowerPC code. In this regard, it is important to understand that some of Mac OS 8/9 still remains as 680x0 code. For example, in Mac OS 8.6, parts of the Control Manager (including, possibly, `TrackControl`) remains in 680x0 code. For the purposes of explanation, the following assumes that, in Mac OS 8/9, `TrackControl` still exists as 680x0 code.

The Mixed Mode Manager

In Mac OS 8/9, the emulator works together with a manager called the **Mixed Mode Manager.** The Mixed Mode Manager manages **mode switches** between code in different **instruction set architectures (ISAs).**

Mode Switches

In Mac OS 8/9, mode switches are required when an application calls a system software function (or, indeed, any other code) that exists in a different ISA. For example:

- When a PowerPC application invokes a system software function that exists only as 680x0 code, a mode switch is required to move from the native environment to the emulator environment. Then, when that system software function completes, another mode switch is required to return from the emulator to the PowerPC environment to allow the PowerPC application to continue executing.

  This situation can occur in a Carbon application running on Mac OS 8/9 because, as previously stated, not all of Mac OS 8/9 has been rewritten as native PowerPC code.

- When a 680x0 application running under the emulator calls a system software

function that exists as native PowerPC code, a mode switch is required to move out of the emulator and into the native PowerPC environment. Then, when that system software function completes, another mode switch is required to return to the emulator and to allow the 680x0 application to continue executing.

This situation cannot occur in a Carbon application running on Mac OS 8/9 because all Carbon applications must be compiled as native PowerPC code.

The Mixed Mode Manager operates transparently to most applications and other types of software, meaning that most **cross-mode calls** (calls to code in a different ISA from the caller's ISA) are detected automatically by the Mixed Mode Manager and handled without intervention by the calling software.

Intervention in Mode Switching

Sometimes, however, executable code needs to interact directly with the Mixed Mode Manager to ensure that a mode switch occurs at the correct time. When writing native PowerPC code, you only have to intervene in the mode-switching process when you execute code whose ISA might be different from the calling code. For example, when you pass the address of your application-defined action function (native PowerPC code) to `TrackControl` (680x0 code), the ISA of the code whose address you are passing is different from the ISA of the function you are passing it to. In such cases, you must ensure that the Mixed Mode Manager is called to make the necessary mode switch. You do this by explicitly signaling

- The type of code you are passing
- The code's calling conventions

Indicating the ISA of a Callback Function-Routine Descriptors

You create a **routine descriptor** for a particular function to indicate the ISA of that function. The first field of a routine descriptor (`goMixedModeTrap`) is an executable 680x0 instruction which invokes the Mixed Mode Manager. The Mixed Mode Manager having been called, it inspects the remaining fields of the routine descriptor to determine whether a mode switch is required. The Mixed Mode Manager is particularly interested in the `routineRecords` field.

The `routineRecords` field is an array of **routine structures,** each element of which describes a single function. In the simplest case, the array of routine structures contains a single element.

The `ISA` and `procInfo` fields are the most important fields in a routine structure:

- ***ISA Field.*** The `ISA` field contains the ISA code of the function being described,

and always contains one of these two constants:

```
kM68kISA    = (ISAType) 0    MC680x0 architecture.
KPowerPCISA = (ISAType) 1    PowerPC Architecture.
```

procInfo Field. The `procInfo` field contains the function's **function information,** including the function's calling conventions and information about the function's parameters.

Creating a Routine Descriptor For a Control Action Function

Using the function `NewControlActionUPP`, you can create a routine descriptor for a control action function as follows, in which `myControlAction` is your application-defined control action function:

```
ControlActionUPP myControlActionUPP;

myControlActionUPP = NewControlActionUPP(myControlAction);
```

Notice that the result returned by `NewControlActionUPP` is of type `ControlActionUPP`. The `UPP` stands for a **universal procedure pointer,** which is defined to be *either* a 680x0 function pointer *or* a pointer to a routine descriptor (hence the term "universal"). Thus, in Mac OS 8/9, the effect of the call to `NewControlActionUPP` depends on whether it is executed in the 680x0 environment or the PowerPC environment:

- In the 680x0 environment, `NewControlActionUPP` simply returns its first parameter-that is, a pointer to your application-defined control action function.
- In the PowerPC environment, `NewControlActionUPP` creates a routine descriptor in your application heap and returns the address of that routine descriptor.

Effect of the Routine Descriptor

Once you have created the routine descriptor, you can later call `TrackControl` like this:

```
TrackControl(myControl,myPoint,myControlActionUPP);
```

In Mac OS 8/9, if your application is a PowerPC application (as will be the case in Carbon), the value passed in the `myControlActionUPP` parameter is not the address of your action function itself, but the address of the routine descriptor. If a 680x0 version of `TrackControl` executes your action function, it begins by executing the instruction in the first field of the routine descriptor. That instruction invokes the Mixed Mode Manager, which inspects the ISA of the action function (contained in the `ISA` field of the routine structure). Since that ISA differs from the ISA of the `TrackControl` function, the

Mixed Mode Manager causes a mode switch. (Of course, if `TrackControl` existed as PowerPC code, the ISAs would be identical, and the Mixed Mode Manager would simply execute the action function without switching modes.)

In short, you solve the problem of indicating a routine's ISA by creating a routine descriptor and by using the address of that routine descriptor (that is, a universal procedure pointer) where you would have used the address of the function (that is, a procedure pointer) in the 680x0 programming environment.

Disposing of Routine Descriptors

Disposing of routine descriptors is only necessary or advisable if you know that you will not be using the descriptor any more during the execution of your application or if you allocate a routine descriptor for temporary use only.

Functions Requiring Routine Descriptors

Some of the typical functions for which you need to create routine descriptors are

Function Type	***Examples Are at the Demonstration Programs Associated With***
Control action functions	Chapters 7, 14, 17 and 21
Event filter functions	Chapters 4, 8, 14 and 21
Apple event handling functions	Chapters 10, 18 and 26
Key filter functions	Chapter 14
Edit text validation functions	Chapter 14
User pane drawing functions	Chapters 14 and 21
User pane activate functions	Chapter 14
Carbon event handlers	Chapters 17 to 26
Carbon event timers	Chapter 17, 18, 21, 23 and 26
Navigation Services event handlers	Chapter 18, 21, 23 and 26
TextEdit click loop functions	Chapter 21
List search functions	Chapter 22

List definition functions Chapter 22

Drag and drop functions (various) Chapter 23

Device loop drawing functions Chapter 25

Universal Procedure Pointers and Carbon

Carbon supports universal procedure pointers transparently. By using the system-supplied universal procedure pointer functions, your application will operate correctly in both the Mac OS 8/9 and Mac OS X environments.

On Mac OS 8/9, the universal procedure pointer creation functions allocate routine descriptors in memory just as you would expect. On Mac OS X, the implementation of universal procedure pointers depends on various factors, including the object file format you choose. Universal procedure pointers will allocate memory if your application is compiled as a CFM binary, but are likely to return a simple procedure pointer if your application is compiled as a Mach-O binary. (All demonstration programs in this book are compiled as CFM binaries so that they will run on both Mac OS 8/9 and Mac OS X. Mach-O binaries only run on Mac OS X.) In Carbon, routine descriptors must be disposed of using the specific disposal function associated with the creation function. For example, routine descriptors created with `NewControlActionUPP` must be disposed of using `DisposeControlActionUPP`. The generic disposal function `DisposeRoutineDescriptor` is not supported in Carbon.

Creation/Disposal Functions Relevant to Demonstration Programs

Creating Routine Descriptors

```
ControlActionUPP             NewControlActionUPP(ControlActionProcPtr userRoutine);
ModalFilterUPP               NewModalFilterUPP(ModalFilterProcPtr userRoutine);
AEEventHandlerUPP            NewAEEventHandlerUPP(AEEventHandlerProcPtr userRoutine);
ControlKeyFilterUPP          NewControlKeyFilterUPP(ControlKeyFilterProcPtr userRoutine);
ControlEditTextValidationUPP NewControlEditTextValidationUPP
(ControlEditTextValidationProcPtr
                             userRoutine);
ControlUserPaneDrawUPP       NewControlUserPaneDrawUPP(ControlUserPaneDrawProcPtr
userRoutine);
ControlUserPaneActivateUPP   NewControlUserPaneActivateUPP(ControlUserPaneActivateProcPtr
                             userRoutine);
TEClickLoopUPP               NewTEClickLoopUPP(TEClickLoopProcPtr userRoutine);
ListSearchUPP                NewListSearchUPP(ListSearchProcPtr userRoutine);
ListDefUPP                   NewListDefUPP(ListDefProcPtr    userRoutine);
DragTrackingHandlerUPP       NewDragTrackingHandlerUPP(DragTrackingHandlerProcPtr
userRoutine);
DragReceiveHandlerUPP        NewDragReceiveHandlerUPP(DragReceiveHandlerProcPtr
userRoutine);
DragSendDataUPP              NewDragSendDataUPP(DragSendDataProcPtr userRoutine);
DragInputUPP                 NewDragInputUPP(DragInputProcPtr userRoutine);
DragDrawingUPP               NewDragDrawingUPP(DragDrawingProcPtr userRoutine);
DeviceLoopDrawingUPP         NewDeviceLoopDrawingUPP(DeviceLoopDrawingProcPtr userRoutine);
```

Disposing of Routine Descriptors

```
void  DisposeControlActionUPP(ControlActionUPP userUPP);
void  DisposeModalFilterUPP(ModalFilterUPP userUPP);
void  DisposeAEEventHandlerUPP(AEEventHandlerUPP userUPP);
void  DisposeControlKeyFilterUPP(ControlKeyFilterUPP userUPP);
void  DisposeControlEditTextValidationUPP(ControlEditTextValidationUPP userUPP);
void  DisposeControlUserPaneDrawUPP(ControlUserPaneDrawUPP userUPP);
void  DisposeControlUserPaneActivateUPP(ControlUserPaneActivateUPP userUPP);
void  DisposeTEClickLoopUPP(TEClickLoopUPP userUPP);
void  DisposeListSearchUPP(ListSearchUPP userUPP);
void  DisposeListDefUPP(ListDefUPP userUPP);
void  DisposeDragTrackingHandlerUPP(DragTrackingHandlerUPP userUPP);
void  DisposeDragReceiveHandlerUPP(DragReceiveHandlerUPP userUPP);
void  DisposeDragSendDataUPP(DragSendDataUPP userUPP);
void  DisposeDragInputUPP(DragInputUPP userUPP);
void  DisposeDragDrawingUPP(DragDrawingUPP userUPP);
void  DisposeDeviceLoopDrawingUPP(DeviceLoopDrawingUPP userUPP);
```

Chapter 6. The Appearance Manager

IN THIS CHAPTER

- History
- The Appearance Manager
- Main Constants, Data Types, and Functions
- Demonstration Program Appearance Listing
- Demonstration Program Appearance Comments

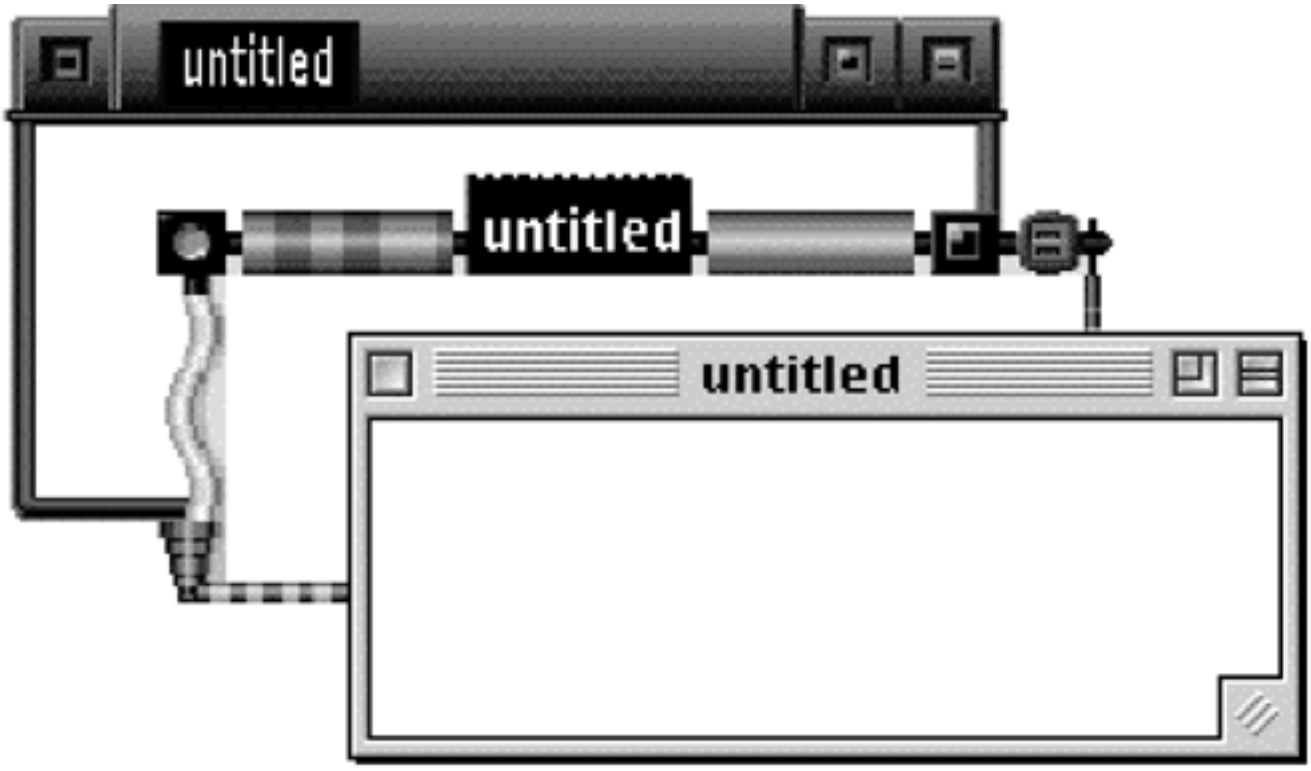
History

The Appearance Manager, which was first introduced with Mac OS 8.0, had implications for the Menu Manager, the Window Manager, the Control Manager, and the Dialog Manager. The relatively minor implications in respect of the Menu Manager and Window Manager were incorporated into Chapter 3 and Chapter 4. The most profound impact of the Appearance Manager, however, has been in the area of user interface objects known as **controls,** which are addressed in Chapters 7 and 14. Accordingly, as a preparation for what is to come, this chapter now formally introduces the Appearance Manager, a component of the system software which, on Mac OS 8/9, represented the most significant improvement in the Macintosh user experience since the introduction of System 7.

Although introduced with Mac OS 8.0, the Appearance Manager's full impact on the Macintosh user experience was not scheduled to be realized until the release of Mac OS 8.5. Mac OS 8.5 was to be the first release to include several switchable **themes,** one of which (the Platinum theme) had, in fact, been included in Mac OS 8.0. The concept of switchable themes was the main driving force behind the creation of the Appearance Manager.

Essentially, a theme was intended to be an interface "look" that spanned all elements of the user interface (windows, menus, dialogs, controls, background colors, alert icons, and so on), tying them together with a certain graphic design. Fig 1 shows the same window as it would have appeared in the three themes originally intended to be included in Mac OS 8.5. If one of these themes had been selected by the user, all elements of the user interface (menus, windows, controls) would have appeared in that theme.

Figure 1. WINDOWS IN THREE THEMES



The two additional themes (High Tech and Gizmo) shown in Fig 1 were included in prerelease versions of Mac OS 8.5; however, prior to final release, these two themes were deleted. The reasons for this decision by Apple remain tantalizingly obscure.

Themes-New Definition

Mac OS 8.5 did, in fact, introduce a theme scheme, though one of an entirely different flavor to that described above. This is reflected in the Appearance control panel introduced with Mac OS 8.5, in which the Platinum *theme* is now referred to as the Platinum *appearance.* An appearance (new definition) is now simply one component of a broader set of user preferences known as a theme (new definition). With the release of Mac OS 8.5, therefore, the term "theme" took on an entirely new meaning.

On Mac OS 8/9, in Mac OS 8.5 and later, an individual theme is a set of user preferences encompassing

- An appearance (which unifies the look of human interface objects such as windows, dialogs, alerts, menus, controls, and so on), together with a highlight color (for selected text) and a variation color (for menus and controls). (As of Mac OS 9.1, Platinum remains the only appearance ever provided by Apple. It is by now certain that this situation will never change.)
- A large system font (for menus and headings), a small system font (for explanatory text and labels), a views font (for lists and icons), and an option to turn anti-aliasing of fonts on screen on or off.
- A desktop picture and desktop pattern.
- Sound preferences relating to opening menus and choosing items, dragging and resizing

windows, interacting with controls, and clicking, dragging, and dropping in the Finder.
- Scrolling preference (smart scrolling on or off) and collapse-window preference (double-click title bar to collapse window on or off).

Theme-Compliance

Another significant terminological change ushered in by Mac OS 8.5 was that, whereas Apple documentation previously spoke of making applications **appearance-compliant,** documentation released following the release of Mac OS 8.5 spoke of making applications **theme-compliant.** It is assumed that the reason for this change is that, while the vast bulk of the measures required to make an application theme-compliant relate to unifying the look of the application's Mac OS 8/9 user interface elements (the province of an appearance), there are additional measures that the application may take, or may have to take:

- In response to the user changing the system and/or views fonts, using the Fonts tab of the Appearance control panel, while the application is running. (This consideration does not apply if the application uses standard human interface elements (that is, system-defined windows, controls, and menus), since the fonts used for these elements automatically change with the theme change. However, some applications may use custom human interface elements and may, for example, draw their own text in a dialog. In such cases, the application must ensure that the fonts used match the corresponding system fonts in the current theme.)
- To cause theme-compliant sounds to accompany, for example, the opening and closing of the application's windows and the manipulation by the user of custom human interface elements.
- To support the proportional scroll boxes the user expects when Smart Scrolling is selected on in the Options tab of the Appearance control panel.

Note

Proportional scroll boxes are scroll boxes which vary in size according to the proportion of the document visible in the window.

The Appearance Manager

The influence of the Appearance Manager is evident to a greater or lesser extent in many chapters of this book and in all of the associated demonstration programs. Among other things, it ensures consistency in the appearance of the standard human interface elements on both Mac OS 8/9 and Mac OS X. It also provides the means to

- Ensure that the appearance of your application's custom human interface elements (if any) is consistent with the Platinum and Aqua "look."
- Draw anti-aliased text on Mac OS X.

Carbon fully supports the Appearance Manager.

New Definition Functions-Mac OS 8/9

To provide a system-wide coordination of look and behavior on Mac OS 8/9, new theme-compliant definition functions were introduced with the Appearance Manager to replace the old pre-Appearance Manager definition functions for menu bars, menus, windows, and controls. In addition, many new theme-compliant control definition functions for new types of controls (slider controls, focus rings, group boxes, and so on) were introduced to obviate the necessity for developers to provide their own.

Colors, Patterns, and Appearance Primitives

The Appearance Manager provides **Appearance primitives,** and the means to set the colors and patterns, needed to draw consistently in the Platinum appearance on Mac OS 8/9 and with the Aqua "look" on Mac OS X. Using these drawing primitives, colors, and patterns makes it easier to create visual entities and custom human interface elements that are consistent with the Platinum appearance and Aqua "look."

Drawing Appearance Primitives

As will become apparent in Chapters 7 and 14, most of the Appearance primitives relate to certain controls. The definition functions for these controls call these primitives when drawing the relevant control. For example, the control definition function for a primary group box calls the primitive `DrawThemePrimaryGroup` to draw the visual representation of that control.

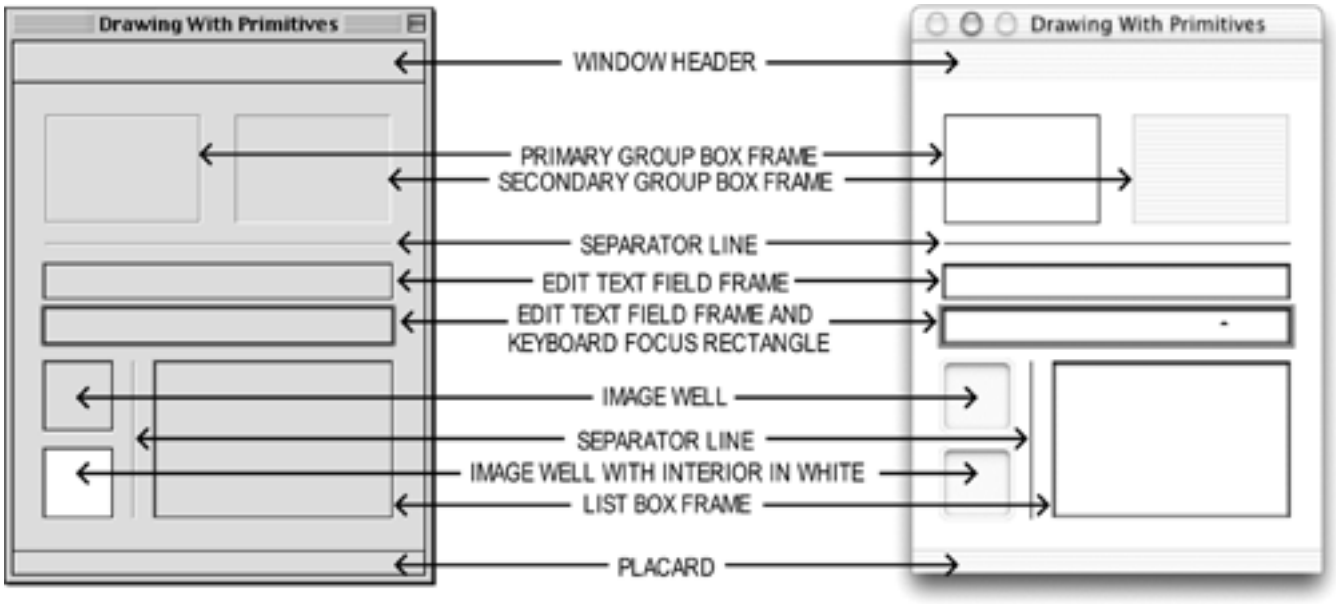
Your application might use these primitives to, for example, draw an image of a placard, window header, or edit text field frame, when you don't want to use a control.

The following are examples of functions that draw Appearance primitives:

Function	*Description*
`DrawThemePrimaryGroup`	Draws a primary group box frame.
`DrawThemeSecondaryGroup`	Draws a secondary group box frame.
`DrawThemeSeparator`	Draws a separator line. The orientation of the rectangle determines how the separator line is drawn. If the rectangle is wider than it is tall, the separator line is horizontal; otherwise it is vertical.
`DrawThemeWindowHeader`	Draws a window header.
`DrawThemePlacard`	Draws a placard.
`DrawThemeEditTextFrame`	Draws an edit text field frame. The rectangle passed in should be the same as the one passed in the function `DrawThemeFocusRect` so you get the correct focus look for your edit text field. You should not use these frames for items other than edit text fields.
`DrawThemeListBoxFrame`	Draws a list box frame. The rectangle passed in should be the same as the one passed into the function `DrawThemeFocusRect` so that you get the correct focus look for your list box.
`DrawThemeFocusRect`	Draws or erases a focus ring around a specified rectangle. To achieve the right look, you should first call `DrawThemeEditTextFrame` or `DrawThemeListBoxFrame` and then call `DrawThemeFocusRect`, passing the same rectangle in the `inRect` parameter. If you use `DrawThemeFocusRect` to erase the focus ring around an edit text field frame or list box frame, you will have to redraw the edit text field frame or list box frame because there is typically an overlap.
`DrawThemeGenericWell`	Draws an image well frame. You can specify that the center of the well be filled with white (Mac OS 8/9).
`DrawThemeFocusRegion`	Draws or erases a focus ring around a specified region.
`DrawThemeTabPane`	Draws a tab-pane.
`DrawThemeTab`	Draws a tab.

Fig 2 shows examples of images drawn in the active mode using the Appearance primitives.

Figure 2. IMAGES DRAWN WITH APPEARANCE DRAWING PRIMITIVES



Draw State Constants

The following constants are passed in the `inState` parameter of the functions that draw Appearance primitives (except `DrawThemeFocusRect` and `DrawThemeFocusRegion`) to specify whether the primitive should be drawn in the active or deactivated mode. (`DrawThemeFocusRect` and `DrawThemeFocusRegion` either draw or erase the focus rectangle depending on whether `true` or `false` is passed in the `inHasFocus` parameter.)

Constant	*Value*	*Description*
`kThemeStateInactive`	`0`	Draw the primitive in the inactive mode
`kThemeStateActive`	`1`	Draw the primitive in the active mode

Another draw state constant (`kThemeStatePressed`) is available to draw certain primitives in the pressed mode; however, the primitives listed above can only be drawn in the active and inactive modes.

Drawing in Colors and Patterns Consistent with the Platinum Appearance and Aqua "Look"

The following functions are those used to draw using colors/patterns consistent with the Platinum appearance and Aqua "look". (Patterns are explained in Chapter 11.) The reference to colors *and* patterns reflects the fact that either a color or a pattern may be used for the drawing.

Function	*Description*

`SetThemeWindowBackground`	Sets the color/pattern that the window background will be repainted to when `PaintOne` is called. This function sets the color/pattern to which the Window Manager will erase the window background. See the Brush Type Constants.
`SetThemeBackground`	Sets an element's background color/pattern to comply with the Platinum appearance/Aqua "look." This function should be called each time you wish to draw an element in a specified brush constant using Appearance Manager draw functions. See the Brush Type Constants.
`SetThemePen`	Sets an element's pen pattern or color to comply with the Platinum appearance/Aqua "look." This function should be called each time you wish to draw an element in a specified brush constant using Appearance Manager draw functions. See the Brush Type Constants.
`SetThemeTextColor`	Sets an element's foreground color for drawing text to comply with the Platinum appearance/Aqua "look." See the Text Color Constants.

Brush Type Constants

The following are examples of constants, of type `ThemeBrush`, which may be passed in the `inBrush` parameter of calls to `SetThemeWindowBackground`, `SetThemeBackground`, and `SetThemePen` to specify colors/patterns for user interface elements. For reasons explained above, these constants can represent either a straight color or a pattern.

Constant	*Description*
`kThemeBrushDialogBackgroundActive`	An active dialog's background color/pattern.

`kThemeBrushDialogBackgroundInactive`	An inactive dialog's background color or pattern.
`kThemeBrushAlertBackgroundActive`	An active alert's background color/pattern.
`kThemeBrushAlertBackgroundInactive`	An inactive alert's background color/pattern.
`kThemeBrushModelessDialogBackgroundActive`	An active modeless dialog's background color/ pattern.
`kThemeBrushModelessDialogBackgroundInactive`	An inactive modeless dialog's background color/ pattern.
`kThemeBrushUtilityWindowBackgroundActive`	An active utility window's background color/pattern.
`kThemeBrushUtilityWindowBackgroundInactive`	An inactive utility window's background color/pattern.
`kThemeBrushListViewSortColumnBackground`	The background color/ pattern of the column upon which a list view is sorted. (Applicable on Mac OS 8/9 only.)
`kThemeBrushListViewBackground`	The background color/ pattern of a list view column that is not being sorted upon. (Applicable on Mac OS 8/9 only.)
`kThemeBrushListViewSeparator`	A list view separator's color/pattern. (Applicable on Mac OS 8/9 only.)
`kThemeBrushDocumentWindowBackground`	A document window's background color/pattern.

Text Color Constants

Constants of type `ThemeTextColor` may be passed in the `inColor` parameter of the function `SetThemeTextColor` to specify theme-compliant text colors for user interface elements in their active, inactive, and highlighted states. Some of these constants are as follows:

Constant	*Description*
`kThemeTextColorWindowHeaderActive`	Text color for active window header.
`kThemeTextColorWindowHeaderInactive`	Text color for inactive window header.
`kThemeTextColorPlacardActive`	Text color for active placard.
`kThemeTextColorPlacardInactive`	Text color for inactive placard.
`kThemeListViewTextColor`	Text color for list view. (Applicable on Mac OS 8/9 only.)

Appearance Manager Text

The Appearance Manager function `UseThemeFont` may be used to set the font for the current graphics port.

Text drawn on Mac OS X using QuickDraw functions such as `DrawString` is not entirely satisfactory. You should therefore use the Appearance Manager function `DrawThemeTextBox` to draw text when your application is running on Mac OS X.

You pass a value of type `ThemeFontID` in the `inFontID` parameter of `UseThemeFont` and `DrawThemeTextBox`. The principal relevant constants are as follows:

Constant	*Font on Mac OS 8/9*	*Font on Mac OS X*
`kThemeSystemFont`	As set in Appearance control panel.	Lucida Grande Regular 13pt
`kThemeEmphasizedSystemFont`	System font, as set in Appearance control panel.	Lucida Grande Bold 13pt
`kThemeSmallSystemFont`	As set in Appearance control panel.	Lucida Grande Regular 11pt
`kThemeSmallEmphasizedSystemFont`	Small system font, as set in Appearance control panel, bold.	Lucida Grande Bold 11pt
`kThemeApplicationFont`	Geneva 12pt.	Lucida Grande Regular 13pt

`kThemeLabelFont`	System font, as set in Appearance control panel	Lucida Grande Regular 10pt

Saving and Setting the Graphics Port Drawing State

Chapter 12 addresses certain measures which need to be taken consequential to the fact that both colors and patterns can be used by the Appearance functions `SetThemeWindowBackground`, `SetThemeBackground`, and `SetThemePen`. These measures have to do with saving, restoring, and normalizing the drawing state of the graphics port. The associated functions are as follows:

Constant	*Description*
`GetThemeDrawingState`	Obtain the drawing state of the current graphics port.
`SetThemeDrawingState`	Set the drawing state of the current graphics port.
`NormalizeThemeDrawingState`	Set the current graphics port to the default drawing state.
`DisposeThemeDrawingState`	Release the memory associated with a reference to a graphics port's drawing state.

Cursor Setting

The Appearance Manager introduced the following cursor-setting functions, the uses of which are addressed in Chapter 13:

Constant	*Description*
`SetThemeCursor`	Sets the cursor.
`SetAnimatedThemeCursor`	Sets an animated cursor.

Getting Menu Bar Height

The Appearance Manager introduced the function `GetThemeMenuBarHeight`. In most instances, the value returned by this function and `GetMBarHeight` are the same. However, when the menu bar is hidden, `GetMBarHeight` produces a value of `0`, whereas `GetThemeMenuBarHeight` still returns the height of the (hidden) menu bar.

Appearance Manager Apple Events

On Mac OS 8/9, your application may need to respond to the user changing the system and/or views fonts using the Fonts tab in the Appearance control panel. Your application is advised of font changes via **Appearance Manager Apple events.** Appearance Manager Apple events are addressed in Chapter 10.

Carbon Note

Prior to CarbonLib 1.1, it was necessary to call `RegisterAppearanceClient` at program launch in order for your application to receive Appearance Manager Apple events. However, in CarbonLib 1.1 and later, the CarbonLib initialization routine calls `RegisterAppearanceClient` on behalf of your application, and there is thus no requirement for your application to call this function.

Main Constants, Data Types, and Functions

Constants

Theme-Compliant Brush Type Constants

```
kThemeBrushDialogBackgroundActive           = 1
kThemeBrushDialogBackgroundInactive         = 2
kThemeBrushAlertBackgroundActive            = 3
kThemeBrushAlertBackgroundInactive          = 4
kThemeBrushModelessDialogBackgroundActive   = 5
kThemeBrushModelessDialogBackgroundInactive = 6
kThemeBrushUtilityWindowBackgroundActive    = 7
kThemeBrushUtilityWindowBackgroundInactive  = 8
kThemeBrushListViewSortColumnBackground     = 9
kThemeBrushListViewBackground               = 10
kThemeBrushListViewSeparator                = 12
kThemeBrushDocumentWindowBackground         = 15
kThemeBrushFinderWindowBackground           = 16
kThemeBrushBlack                            = -1
kThemeBrushWhite                            = -2
```

Theme-Compliant Text Color Constants

```
kThemeTextColorWindowHeaderActive           = 7
kThemeTextColorWindowHeaderInactive         = 8
kThemeTextColorPlacardActive                = 9
kThemeTextColorPlacardInactive              = 10
kThemeTextColorPlacardPressed               = 11
kThemeTextColorListView                     = 22
kThemeTextColorBlack                        = -1
kThemeTextColorWhite                        = -2
```

Theme-Compliant Draw State Constants (For Primitives)

```
kThemeStateInactive                         = 0
kThemeStateActive                           = 1
kThemeStatePressed                          = 2
```

Theme Cursor Constants

```
kThemeArrowCursor                           = 0
kThemeCopyArrowCursor                       = 1
kThemeAliasArrowCursor                      = 2
kThemeContextualMenuArrowCursor             = 3
kThemeIBeamCursor                           = 4
kThemeCrossCursor                           = 5
kThemePlusCursor                            = 6
kThemeWatchCursor                           = 7    Can animate
kThemeClosedHandCursor                      = 8
kThemeOpenHandCursor                        = 9
kThemePointingHandCursor                    = 10
kThemeCountingUpHandCursor                  = 11   Can animate
kThemeCountingDownHandCursor                = 12   Can animate
kThemeCountingUpAndDownHandCursor           = 13   Can animate
kThemeSpinningCursor                        = 14   Can animate
kThemeResizeLeftCursor                      = 15
kThemeResizeRightCursor                     = 16
kThemeResizeLeftRightCursor                 = 17
```

Font Constants

```
kThemeSystemFont                        = 0
kThemeSmallSystemFont                   = 1
kThemeSmallEmphasizedSystemFont         = 2
kThemeViewsFont                         = 3
kThemeEmphasizedSystemFont              = 4
kThemeApplicationFont                   = 5
kThemeLabelFont                         = 6
kThemeCurrentPortFont                   = 200
```

Data Types

```
typedef UInt32 ThemeDrawState;
typedef SInt16 ThemeBrush;
typedef SInt16 ThemeTextColor;
typedef UInt32 ThemeCursor;
typedef struct OpaqueThemeDrawingState* ThemeDrawingState;
```

Functions

Drawing Appearance Primitives

```
OSStatus  DrawThemeWindowHeader(const Rect *inRect,ThemeDrawState inState);
OSStatus  DrawThemeWindowListViewHeader(const Rect *inRect,ThemeDrawState inState);
OSStatus  DrawThemePlacard(const Rect *inRect,ThemeDrawState inState);
OSStatus  DrawThemeEditTextFrame(const Rect *inRect,ThemeDrawState inState);
OSStatus  DrawThemeListBoxFrame(const Rect *inRect,ThemeDrawState inState);
OSStatus  DrawThemeFocusRect(const Rect *inRect,Boolean inHasFocus);
OSStatus  DrawThemePrimaryGroup(const Rect *inRect,ThemeDrawState i
nState);
OSStatus  DrawThemeSecondaryGroup(const Rect *inRect,ThemeDrawState inState);
OSStatus  DrawThemeSeparator(const Rect *inRect,ThemeDrawState inState);
OSStatus  DrawThemeModelessDialogFrame(const Rect *inRect,ThemeDrawState inState);
OSStatus  DrawThemeGenericWell(const Rect *inRect,ThemeDrawState inState,
          Boolean inFillCenter);
OSStatus  DrawThemeFocusRegion(RgnHandle inRegion,Boolean inHasFocus);
OSStatus  DrawThemeTab(const Rect *inRect,ThemeTabStyle inStyle,ThemeTabDirection
inDirection,
          ThemeTabTitleDrawUPP labelProc,UInt32 userData);
OSStatus  DrawThemeTabPane(const Rect *inRect,ThemeDrawState inState);
```

Drawing in Colors/Patterns Consistent with Platinum/Aqua

```
OSStatus  SetThemeWindowBackground(WindowPtr inWindow,ThemeBrush inBrush,Boolean inUpdate);
OSStatus  SetThemeBackground(ThemeBrush inBrush,SInt16 inDepth,Boolean inIsColorDevice);
OSStatus  SetThemePen(ThemeBrush inBrush,SInt16 inDepth,Boolean inIsColorDevice);
OSStatus  SetThemeTextColor(ThemeTextColor inColor,SInt16 inDepth,Boolean inIsColorDevice);
```

Setting and Getting the Graphics Port Font

```
OSStatus  UseThemeFont(ThemeFontID inFontID,ScriptCode inScript);
OSStatus  GetThemeFont(ThemeFontID inFontID,ScriptCode inScript,Str255 outFontName,
           SInt16 *outFontSize,Style *outStyle);
```

Drawing Text

```
OSStatus  DrawThemeTextBox(CFStringRef inString,ThemeFontID inFontID,ThemeDrawState inState,
          Boolean inWrapToWidth,const Rect *inBoundingBox,SInt16 inJust,void *inContext);
OSStatus  TruncateThemeText(CFMutableStringRef inString,ThemeFontID inFontID,
          ThemeDrawState inState,SInt16 inPixelWidthLimit,TruncCode inTruncWhere,
          Boolean *outTruncated);
OSStatus  GetThemeTextDimensions(CFStringRef inString,ThemeFontID inFontID,
```

```
          ThemeDrawState inState,Boolean inWrapToWidth,Point *ioBounds,SInt16 *outBaseline);
OSStatus  GetThemeTextShadowOutset(ThemeFontID inFontID,ThemeDrawState inState,
          Rect * outOutset);
```

Saving and Setting the Graphics Port Drawing State

```
OSStatus  NormalizeThemeDrawingState(void);
OSStatus  GetThemeDrawingState(ThemeDrawingState *outState);
OSStatus  SetThemeDrawingState(ThemeDrawingState inState,Boolean inDisposeNow);
OSStatus  DisposeThemeDrawingState(ThemeDrawingState inState);
```

Setting Appearance Cursors

```
OSStatus  SetThemeCursor(ThemeCursor inCursor);
OSStatus  SetAnimatedThemeCursor(ThemeCursorinCursor,UInt32 inAnimationStep);
```

Getting Menu Bar Height

```
OSStatus  GetThemeMenuBarHeight(SInt16 *outHeight);
```

Demonstration Program Appearance Listing

```
// ****************************************************************************************
// Appearance.c                                                      CLASSIC EVENT MODEL
// ****************************************************************************************
//
// This program opens two kWindowDocumentProc windows containing:
//
// •  In the first window:
//
//    •  On Mac OS 8/9, a theme-compliant list view.
//
//    •  On Mac OS X, some text drawn with Appearance Manager functions.
//
// •  In the second window, various images drawn with Appearance primitives and, on Mac OS
//    8/9, text drawn in the window header in the correct Appearance color.
//
// Two of the images in the second window are edit text field frames and one is a list box
// frame. At any one time, one of these will have a keyboard focus frame drawn around it.
// Clicking in one of the other frames will move the keyboard focus frame to that frame.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  An 'MBAR' resource, and 'MENU' resources for Apple, File, and Edit menus (preload,
//    non-purgeable).
//
// •  Two 'WIND' resources (purgeable) (initially not visible).
//
// •  A 'STR#' list resource (purgeable) containing text drawn in the first window.
//
// •  'hrct' and 'hwin' resources (both purgeable), which provide help balloons describing the
//     contents of the windows (Mac OS 8/9).
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents,  canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// ****************************************************************************************

// ..........................................................................................................................................................................................
includes

#include <Carbon.h>

// ..........................................................................................................................................................................................
defines

#define rMenubar          128
#define rNewWindow1       128
#define rNewWindow2       129
#define mAppleApplication 128
#define  iAbout           1
#define mFile             129
#define  iQuit            12
#define sDescription      128
#define MAX_UINT32        0xFFFFFFFF

// ..................................................................................................................................................... global variables

Boolean   gRunningOnX = false;
Boolean   gDone;
Boolean   gInBackground;
WindowRef gWindowRef1, gWindowRef2;
SInt16    gPixelDepth;
Boolean   gIsColorDevice = false;
Rect      gCurrentRect;
Rect      gEditText1Rect = {  141, 20, 162, 239 } ;
Rect      gEditText2Rect = {  169, 20, 190, 239 } ;
Rect      gListBoxRect   = {  203, 90, 300, 239 } ;
// ..................................................................................................................................................... function prototypes

void  main                       (void);
void  doPreliminaries            (void);
OSErr quitAppEventHandler        (AppleEvent *,AppleEvent *,SInt32);
void  doEvents                   (EventRecord *);
void  doUpdate                   (EventRecord *);
void  doActivate                 (EventRecord *);
void  doActivateWindow           (WindowRef,Boolean);
void  doOSEvent                  (EventRecord *);
void  doDrawAppearancePrimitives  (ThemeDrawState);
void  doDrawThemeCompliantTextOn9 (WindowRef,ThemeDrawState);
void  doDrawListViewOn9          (WindowRef);
void  doDrawThemeTextOnX         (WindowRef,ThemeDrawState);
void  doChangeKeyBoardFocus      (Point);
```

```
void  doGetDepthAndDevice        (void);

// ********************************************************************************** main

void  main(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  SInt16        fontNum;
  EventRecord   EventStructure;

  // ..................................................................................................................................................................................... do preliminaries

  doPreliminaries();

  // ..................................................................................................................................................................... set up menu bar and menus

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    ExitToShell();
  SetMenuBar(menubarHdl);
  DrawMenuBar();

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
      DisableMenuItem(menuRef,0);
    }

    gRunningOnX = true;
  }
  // ..................................................................................... open first window, set font and font size, show window

  if(!(gWindowRef1 = GetNewCWindow(rNewWindow1,NULL,(WindowRef)-1)))
    ExitToShell();

  SetPortWindowPort(gWindowRef1);
  if(!gRunningOnX)
    UseThemeFont(kThemeSmallSystemFont,smSystemScript);
  else
  {
    GetFNum("\pAmerican Typewriter",&fontNum);
    TextFont(fontNum);
    TextSize(11);
  }

  if(!gRunningOnX)
    SetWTitle(gWindowRef1,"\pList Views");
  else
    SetWTitle(gWindowRef1,"\pTheme Text");

  ShowWindow(gWindowRef1);

  // ............................................... open second window, set font, set background color/pattern, show window

  if(!(gWindowRef2 = GetNewCWindow(rNewWindow2,NULL,(WindowRef)-1)))
    ExitToShell();

  SetPortWindowPort(gWindowRef2);
  UseThemeFont(kThemeSmallSystemFont,smSystemScript);

  SetThemeWindowBackground(gWindowRef2,kThemeBrushDialogBackgroundActive,false);

  ShowWindow(gWindowRef2);

  // .................. get pixel depth and whether color device for certain Appearance Manager functions

  if(!gRunningOnX)
    doGetDepthAndDevice();

// ........................................... set top edit text field rectangle as target for initial keyboard focus frame

  gCurrentRect = gEditText1Rect;

  // ..................................................................................................................................................................... enter eventLoop

  gDone = false;

  while(!gDone)
  {
    if(WaitNextEvent(everyEvent,&EventStructure,MAX_UINT32,NULL))
      doEvents(&EventStructure);
  }
```

```
}
// ********************************************************************** doPreliminaries

void  doPreliminaries(void)
{
  OSErr osError;

  MoreMasterPointers(32);
  InitCursor();
  FlushEvents(everyEvent,0);

  osError = AEInstallEventHandler(kCoreEventClass,kAEQuitApplication,
                            NewAEEventHandlerUPP((AEEventHandlerProcPtr) quitAppEventHandler),
                            0L,false);
  if(osError != noErr)
    ExitToShell();
}

// ********************************************************************** doQuitAppEvent

OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  OSErr    osError;
  DescType returnedType;
  Size     actualSize;

  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,NULL,0,
                              &actualSize);

  if(osError == errAEDescNotFound)
  {
    gDone = true;
    osError = noErr;
  }
  else if(osError == noErr)
    osError = errAEParamMissed;

  return osError;
}

// ********************************************************************************** doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
  SInt32         menuChoice;
  MenuID         menuID;
  MenuItemIndex  menuItem;
  WindowPartCode partCode;
  WindowRef      windowRef;

  switch(eventStrucPtr->what)
  {
    case kHighLevelEvent:
      AEProcessAppleEvent(eventStrucPtr);
      break;
    case keyDown:
      if((eventStrucPtr->modifiers & cmdKey) != 0)
      {
        menuChoice = MenuEvent(eventStrucPtr);
        menuID = HiWord(menuChoice);
        menuItem = LoWord(menuChoice);
        if(menuID == mFile && menuItem  == iQuit)
          gDone = true;
      }
      break;

    case mouseDown:
      if(partCode = FindWindow(eventStrucPtr->where,&windowRef))
      {
        switch(partCode)
        {
          case inMenuBar:
            menuChoice = MenuSelect(eventStrucPtr->where);
            menuID = HiWord(menuChoice);
            menuItem = LoWord(menuChoice);

            if(menuID == 0)
              return;

            switch(menuID)
            {
              case mAppleApplication:
                if(menuItem == iAbout)
                {
                  SysBeep(10);
                  HiliteMenu(0);
                }
                break;
```

```
            case mFile:
              if(menuItem == iQuit)
                gDone = true;
              break;
            }
            break;

          case inContent:
            if(windowRef != FrontWindow())
              SelectWindow(windowRef);
            else
            {
              if(FrontWindow() == gWindowRef2)
              {
                SetPortWindowPort(gWindowRef2);
                doChangeKeyBoardFocus(eventStrucPtr->where);
              }
            }
            break;
          case inDrag:
            DragWindow(windowRef,eventStrucPtr->where,NULL);
            break;
        }
      }
      break;

    case updateEvt:
      doUpdate(eventStrucPtr);
      break;

    case activateEvt:
      doActivate(eventStrucPtr);
      break;

    case osEvt:
      doOSEvent(eventStrucPtr);
      break;
  }
}

// ********************************************************************************** doUpdate

void  doUpdate(EventRecord *eventStrucPtr)
{
  WindowRef windowRef;

  windowRef = (WindowRef) eventStrucPtr->message;

  BeginUpdate(windowRef);

  SetPortWindowPort(windowRef);

  if(windowRef == gWindowRef2)
  {
    if(gWindowRef2 == FrontWindow() && !gInBackground)
    {
      doDrawAppearancePrimitives(kThemeStateActive);
      DrawThemeFocusRect(&gCurrentRect,true);
      if(!gRunningOnX)
        doDrawThemeCompliantTextOn9(windowRef,kThemeStateActive);
    }
    else
    {
      doDrawAppearancePrimitives(kThemeStateInactive);
      if(!gRunningOnX)
        doDrawThemeCompliantTextOn9(windowRef,kThemeStateInactive);
    }
  }

  if(windowRef == gWindowRef1)
  {
    if(!gRunningOnX)
      doDrawListViewOn9(windowRef);
  }
  EndUpdxate(windowRef);
}

// ********************************************************************************** doActivate

void  doActivate(EventRecord *eventStrucPtr)
{
  WindowRef windowRef;
  Boolean   becomingActive;

  windowRef = (WindowRef) eventStrucPtr->message;
  becomingActive = ((eventStrucPtr->modifiers & activeFlag) == activeFlag);
  doActivateWindow(windowRef,becomingActive);
}
```

```
// ********************************************************************** doActivateWindow

void  doActivateWindow(WindowRef windowRef,Boolean becomingActive)
{
  if(windowRef == gWindowRef2)
  {
    SetPortWindowPort(gWindowRef2);
    doDrawAppearancePrimitives(becomingActive);
    DrawThemeFocusRect(&gCurrentRect,becomingActive);

    if(!gRunningOnX)
      doDrawThemeCompliantTextOn9(windowRef,becomingActive);

  }
  else if(windowRef == gWindowRef1 && gRunningOnX)
  {
    SetPortWindowPort(gWindowRef1);
    doDrawThemeTextOnX(windowRef,becomingActive);
  }
}

// ************************************************************************** doOSEvent

void  doOSEvent(EventRecord *eventStrucPtr)
{
  switch((eventStrucPtr->message >> 24) & 0x000000FF)
  {
    case suspendResumeMessage:
      if((eventStrucPtr->message & resumeFlag) == 1)
      {
        SetThemeCursor(kThemeArrowCursor);
        gInBackground = false;
      }
      else
        gInBackground = true;
      break;
  }
}
// ********************************************************** doDrawAppearancePrimitives

void  doDrawAppearancePrimitives(ThemeDrawState inState)
{
  Rect theRect;

  if(gRunningOnX)
  {
    GetWindowPortBounds(gWindowRef2,&theRect);
    EraseRect(&theRect);
  }

  SetRect(&theRect,-1,-1,261,26);
  DrawThemeWindowHeader(&theRect,inState);

  SetRect(&theRect,20,46,119,115);
  DrawThemePrimaryGroup(&theRect,inState);

  SetRect(&theRect,140,46,239,115);
  DrawThemeSecondaryGroup(&theRect,inState);

  SetRect(&theRect,20,127,240,128);
  DrawThemeSeparator(&theRect,inState);

  DrawThemeEditTextFrame(&gEditText1Rect,inState);

  DrawThemeEditTextFrame(&gEditText2Rect,inState);

  SetRect(&theRect,20,203,62,245);
  DrawThemeGenericWell(&theRect,inState,false);

  SetRect(&theRect,20,258,62,300);
  DrawThemeGenericWell(&theRect,inState,true);

  SetRect(&theRect,75,202,76,302);
  DrawThemeSeparator(&theRect,inState);

  DrawThemeListBoxFrame(&gListBoxRect,inState);

  SetRect(&theRect,-1,321,261,337);
  DrawThemePlacard(&theRect,inState);
}

// ********************************************************** doDrawThemeCompliantTextOn9

void  doDrawThemeCompliantTextOn9(WindowRef windowRef,ThemeDrawState inState)
{
  SInt16 windowWidth, stringWidth;
  Rect   portRect;
  Str255 message = "\pBalloon help is available";
```

```
  if(inState == kThemeStateActive)
    SetThemeTextColor(kThemeTextColorWindowHeaderActive,gPixelDepth,gIsColorDevice);
  else
    SetThemeTextColor(kThemeTextColorWindowHeaderInactive,gPixelDepth,gIsColorDevice);
  GetWindowPortBounds(windowRef,&portRect);
  windowWidth = portRect.right - portRect.left;
  stringWidth = StringWidth(message);
  MoveTo((windowWidth / 2) - (stringWidth / 2), 17);
  DrawString(message);
}

// ********************************************************************** doDrawListViewOn9

void  doDrawListViewOn9(WindowRef windowRef)
{
  Rect   theRect;
  SInt16 a;

  GetWindowPortBounds(windowRef,&theRect);

  SetThemeBackground(kThemeBrushListViewBackground,gPixelDepth,gIsColorDevice);
  EraseRect(&theRect);

  theRect.left += 130;

  SetThemeBackground(kThemeBrushListViewSortColumnBackground,gPixelDepth,gIsColorDevice);
  EraseRect(&theRect);

  SetThemePen(kThemeBrushListViewSeparator,gPixelDepth,gIsColorDevice);

  GetWindowPortBounds(windowRef,&theRect);

  for(a=theRect.top;a<=theRect.bottom;a+=18)
  {
    MoveTo(theRect.left,a);
    LineTo(theRect.right - 1,a);
  }

  SetThemeTextColor(kThemeTextColorListView,gPixelDepth,gIsColorDevice);

  for(a=theRect.top;a<=theRect.bottom +18;a+=18)
  {
    MoveTo(theRect.left,a - 5);
    DrawString("\p   List View Background        List View Sort Column");
  }
}

// ********************************************************************** doDrawThemeTextOnX

void  doDrawThemeTextOnX(WindowRef windowRef,ThemeDrawState inState)
{
  Rect        portRect;
  Str255      theString;
  CFStringRef stringRef;

  GetWindowPortBounds(windowRef,&portRect);
  EraseRect(&portRect);
  if(inState == kThemeStateActive)
    TextMode(srcOr);
  else
    TextMode(grayishTextOr);

  SetRect(&portRect,portRect.left,30,portRect.right,50);
  DrawThemeTextBox(CFSTR("System Font"),kThemeSystemFont,inState,true,
                   &portRect,teJustCenter,NULL);
  SetRect(&portRect,portRect.left,60,portRect.right,80);
  DrawThemeTextBox(CFSTR("Emphasized System Font"),kThemeEmphasizedSystemFont,inState,true,
                   &portRect,teJustCenter,NULL);
  SetRect(&portRect,portRect.left,90,portRect.right,105);
  DrawThemeTextBox(CFSTR("Small System Font"),kThemeSmallSystemFont,inState,true,
                   &portRect,teJustCenter,NULL);
  SetRect(&portRect,portRect.left,120,portRect.right,135);
  DrawThemeTextBox(CFSTR("Small Emphasized System Font"),kThemeSmallEmphasizedSystemFont,
                   inState,true,&portRect,teJustCenter,NULL);
  SetRect(&portRect,portRect.left,150,portRect.right,170);
  DrawThemeTextBox(CFSTR("Application Font"),kThemeApplicationFont,inState,true,
                   &portRect,teJustCenter,NULL);
  SetRect(&portRect,portRect.left,180,portRect.right,190);
  DrawThemeTextBox(CFSTR("Label Font"),kThemeLabelFont,inState,true,
                   &portRect,teJustCenter,NULL);

  GetIndString(theString,sDescription,1);
  stringRef = CFStringCreateWithPascalString(NULL,theString,CFStringGetSystemEncoding());
  SetRect(&portRect,portRect.left + 20,210,portRect.right - 20,300);
  DrawThemeTextBox(stringRef,kThemeCurrentPortFont,inState,true,&portRect,teJustCenter,NULL);

  if(stringRef != NULL)
    CFRelease(stringRef);
}
```

```
// ********************************************************************* doChangeKeyBoardFocus

void  doChangeKeyBoardFocus(Point mouseXY)
{
  Rect portRect;

  DrawThemeFocusRect(&gCurrentRect,false);
  DrawThemeEditTextFrame(&gCurrentRect,kThemeStateActive);
  SetPortWindowPort(gWindowRef2);
  GlobalToLocal(&mouseXY);

  if(PtInRect(mouseXY,&gEditText1Rect))
    gCurrentRect = gEditText1Rect;
  else if(PtInRect(mouseXY,&gEditText2Rect))
    gCurrentRect = gEditText2Rect;
  else if(PtInRect(mouseXY,&gListBoxRect))
    gCurrentRect = gListBoxRect;

  GetWindowPortBounds(gWindowRef2,&portRect);
  InvalWindowRect(gWindowRef2,&portRect);
}
// ********************************************************************* doGetDepthAndDevice

void doGetDepthAndDevice(void)
{
  GDHandle deviceHdl;

  deviceHdl = GetMainDevice();
  gPixelDepth = (*(*deviceHdl)->gdPMap)->pixelSize;
  if(((1 << gdDevType) & (*deviceHdl)->gdFlags) != 0)
    gIsColorDevice = true;
}

// ********************************************************************************************
```

Demonstration Program Appearance Comments

When this program is run, the user should

- With the "Drawing with Primitives" window frontmost, click in the edit text frame not currently outlined with the keyboard focus frame, or in the list box frame, so as to move the keyboard focus frame to that rectangle.

- Click on the desktop to send the application to the background and note the changed appearance of the frames and text in the "Drawing with Primitives" window. On Mac OS 8/9, note also that there is no change to the appearance of the content region of the "List Views" window. On Mac OS X, note the changed appearance of the text in the "Theme Text" window. Click on the "Drawing with Primitives" window to bring the application to the foreground with that window active, noting the changed appearance of the frames and text. Click on the "Theme Text" window to make it active and note the changed appearance of the text.

- On Mac OS 8/9, Choose Show Balloons from the Help menu and move the cursor over the frames in the window titled "Drawing with Primitives" window (when active), and the left and right sides of the window titled "List Views" (when active), noting the descriptions in the balloons.

In the following, reference is made to graphics devices and pixel depth. Graphics devices and pixel depth are explained in Chapter 11.

Global Variables

gWindowRef1 and gWindowRef2 will be assigned references to window objects.

`gPixelDepth` will be assigned the pixel depth of the main device (screen). `gIsColorDevice` will be assigned true if the graphics device is a color device and false if it is a monochrome device. The values in these two variables are required by certain Appearance Manager functions.

`gCurrentRect` will be assigned the rectangle which is to be the current target for the keyboard focus frame. `gEditText1Rect`, `gEditText2Rect`, and `gListBoxRect` are assigned the rectangles for the two edit text frames and the list box frame.

main

After each window is created, its graphics port is set as the current port before the port's font is set. For the first window's graphics port, if the program is running on Mac OS 8/9, the Appearance Manager function `UseThemeFont` is called to set the font to the small system font. Otherwise the Font Manager function `GetFNum` is called to get the font number for American Typewriter, which is then set as the port's font, at 11 points, by the QuickDraw functions TextFont and TextSize. For the second window's graphics port, `UseThemeFont` is called to set the font to the small system font.

If the program is running on OS 8/9, `SetThemeWindowBackground` is called to set a theme-compliant color/pattern for the "Drawing with Primitives" window's content area. This means that the content area will be automatically repainted with that color/pattern when required with no further assistance from the application. When false is passed in the third parameter, the content region of the window is not invalidated. (Passing true in this instance is not appropriate because the window is not yet showing.)

If the program is running on OS 8/9, `doGetDepthAndDevice` is called to determine the current pixel depth of the graphics port, and whether the current graphics device is a color device, and assign the results to the global variables `gPixelDepth` and `gIsColorDevice`. These values are required by certain Appearance Manager functions which, in this program, are not called if the program is running on Mac OS X.

The next line sets the initial target for the keyboard focus frame. This is the rectangle used by the first edit text field frame.

doEvents

At the `mouseDown` case, the `inContent` case within the `partCode` switch is of relevance to the demonstration.

If the mouse-down was within the content region of a window, and if that window is not the front window, `SelectWindow` is called to bring that window to the front and activate it.

However, if the window is the front window, and if that window is the "Drawing with Primitives" window, that window's graphics port is set as the current graphics port for drawing, and `doChangeKeyBoardFocus` is called. That function determines whether the mouse-down was within one of the edit text field frames or the list box frame, and moves the keyboard focus if necessary.

doUpdate

Within the `doUpdate` function, if the window to which the update event relates is the "Drawing with Primitives" window, if that window is currently the front window, and if the

application is not currently in the background:

- Functions are called to draw the primitives and, on Mac OS 8/9 only, the window header text in the active mode.
- `DrawThemeFocusRect` is called to draw the keyboard focus frame using the rectangle currently assigned to the global variable `gCurrentRect`.

If, however, the "Drawing with Primitives" window is not the front window, the same calls are made but with the primitives and, on Mac OS 8/9 only, text being drawn in the inactive mode. Note that no call is required to erase the keyboard focus frame because this will already have been erased when the window was deactivated.

If the window to which the update event relates is the "List Views" window, `doDrawListViewOn9` is called to draw the window's content area. Note that, for this window, there is no differentiation between active and inactive modes. This is because, for list views, the same brush type constants are used regardless of whether the window is active or inactive.

doActivateWindow

When an activate event is received for the "Drawing with Primitives" window, functions are called to draw the primitives and, on Mac OS 8/9 only, the window header text. In addition, an Appearance Manager function which draws and erases the keyboard focus rectangle is called. The value passed in the `becomingActive` parameter of these calls ensures ensure that, firstly, the primitives and text are drawn in the appropriate mode and, secondly, the keyboard focus frame is either drawn or erased, depending on whether the window is coming to the front or being sent to the back.

If the activate event is for the first window and the program is running on Mac OS X, `doDrawThemeTextOnX` is called to draw some text in the window in either the active or inactive mode.

doDrawAppearancePrimitives

`doDrawAppearancePrimitives` uses Appearance Manager functions for drawing Appearance primitives, and is called to draw the various frames in the "Drawing with Primitives" window. The mode in which the primitives are drawn (active or inactive) is determined by the Boolean value passed in the `inState` parameter.

doDrawThemeCompliantTextOn9

`doDrawThemeCompliantTextOn9`, which is called only if the program is running on Mac OS 8/9, draw some advisory text in the window header of the "Drawing with Primitives" window. The QuickDraw drawing function `DrawString` does the drawing;

however, before the drawing begins, the Appearance Manager function SetThemeTextColor is used to set the foreground color for drawing text, in either the active or inactive modes, so as to comply with the Platinum appearance.

At the first two lines, if "Drawing with Primitives" is the active window, SetThemeTextColor is called with the kThemeTextColorWindowHeaderActive text color constant passed in the first parameter. At the next two lines, if the window is inactive, SetThemeTextColor is called with kThemeTextColorWindowHeaderInactive passed in the first parameter. Note that SetThemeTextColor requires the pixel depth of the graphics port, and whether the graphics device is a color device or a monochrome device, passed in the second and third parameters.

The next four lines simply adjust QuickDraw's pen location so that the text is drawn centered laterally in the window header frame. The call to DrawString draws the specified text.

doDrawListViewOn9

doDrawListViewOn9, which is called only if the program is running on Mac OS 8/9, draws a list view background in the specified window.

The first line copies the window's port rectangle to a local variable of type Rect.

The call to SetThemeBackground sets the background color/pattern to the color/pattern represented by the theme-compliant brush type constant kThemeBrushListViewBackground. The QuickDraw function EraseRect fills the whole of the port rectangle with this color/pattern.

The next line adjusts the Rect variable's left field so that the rectangle now represents the right half of the port rectangle. The same drawing process is then repeated, but this time with kThemeBrushListViewSortColumnBackground passed in the first parameter of the SetThemeBackground call.

SetThemePen is then called with the color/pattern represented by the constant kThemeBrushListViewSeparator passed in the first parameter. The rectangle for drawing is then expanded to equate with the port rectangle before the following five lines draw one-pixel-wide horizontal lines, at 18-pixel intervals, from the top to the bottom of the port rectangle.

Finally, some text is drawn in the list view in the theme-compliant color for list views. SetThemeTextColor is called with the kThemeTextColorListView passed in, following which a for loop draws some text, at 18-pixel intervals, from the top to the bottom of the port rectangle.

doDrawThemeTextOnX

`doDrawThemeTextOnX` is called only if the program is running on Mac OS X. It draws anti-aliased text in the first window.

`GetWindowPortBounds` is called to copy the port rectangle to a local variable of type `Rect`. `EraseRect` is then called to erase the port rectangle, a necessary precursor to drawing over existing anti-aliased text on Mac OS X using the Appearance Manager function `DrawThemeTextBox`.

As was done in the function `doDrawThemeCompliantTextOn9`, `SetThemeTextColor` could be used here to set the text color according to the value received in the `inState` formal parameter. However, in this instance the alternative of calling `TextMode` is used. The so-called transfer modes passed in the calls to `TextMode` are explained in Chapter 12. `srcOr` is the default transfer mode for text, and causes the color of the glyph (the visual representation of a character) to be determined by the graphics port's foreground color. The non-standard mode `grayishTextOr` is used to draw text in the deactivated state.

Before each call to `DrawThemeTextBox`, `SetRect` is called to adjust the top and bottom fields of the `Rect` variable `portRect`. This controls the vertical positioning of the text in the window, being passed in `DrawThemeTextBox`'s `inBoundingBox` parameter. teJustCenter is passed in `DrawThemeTextBox`'s `inJust` parameter to cause the text to be center-justified within the rectangle. The Appearance Manager constants passed in the `inFontID` parameter determine the size and style of the drawn text.

At the last block, a string is retrieved from a `'STR#'` Resource. After being converted to a `CFString`, that string is drawn by `DrawThemeTextBox` in the bottom of the window. Note that `kThemeCurrentPort` passed in the `inFontID` parameter so as to cause the text to be drawn using the window's graphics port font, which was set in main.

doChangeKeyBoardFocus

`doChangeKeyBoardFocus` is called when a mouse-down occurs in the content region of the "Drawing with Primitives" window.

At the first two lines, Appearance Manager functions are used to, firstly, erase the keyboard focus frame from the rectangle around which it is currently drawn and, secondly, redraw an edit text field frame around that rectangle.

The call to `GlobalToLocal` converts the coordinates of the mouse-down to the local coordinates required by the following calls to `PtInRect`. `PtInRect` returns true if the mouse-down is within the rectangle passed in the second parameter. If one of the calls to

`PtInRect` returns true, that rectangle is made the current rectangle for keyboard focus by assigning it to the global variable `gCurrentRect`.

The call to `InvalWindowRect` ensures that the keyboard focus frame will be drawn by the call to `DrawThemeFocusRect` in the function `doUpdate`.

doGetDepthAndDevice

`doGetDepthAndDevice` determines the pixel depth of the graphics port, and whether the graphics device is a color device or a monochrome device, and assigns the results to two global variables. This information is required by the Appearance Manager functions SetThemeTextColor, `SetThemeBackground`, and `SetThemePen`.

Chapter 7. Introduction to Controls

IN THIS CHAPTER

- Introduction
- Standard Controls
- The Control Definition Function
- **The Basic Controls, Primary Group Boxes, and User Panes**
- Activating, Deactivating, Hiding, Showing, Enabling, and Disabling Controls
- Visual Feedback from the Basic Controls
- Embedding Controls
- Getting and Setting Control Data
- The Control Object
- Creating Controls
- Updating, Moving, and Removing Controls
- Handling Mouse Events in Controls
- **Determining and Changing Control Settings**
- Moving and Resizing Scroll Bars
- **Scrolling Operations with Scroll Bars**
- Small Versions of Controls
- **Main Control Manager Constants**
- Demonstration Program Controls1 Listing
- Demonstration Program Controls1 Comments
- Demonstration Program Controls2 Listing
- Demonstration Program Controls2 Comments

Introduction

On Mac OS 8/9, prior to the introduction of Mac OS 8 and the Appearance Manager, the system software provided for only a limited range of **controls** (specifically, buttons, checkboxes, radio buttons, pop-up menus, and scroll bars) and the creation and handling of these desktop objects was relatively simple and straightforward. Mac OS 8 and the Appearance Manager, however, ushered in a very wide range of additional controls, extended the capabilities of the old controls, and provided a generally richer control environment. The result is that the subject of controls is now considerably more involved; accordingly, this chapter constitutes an introduction to controls only and addresses only the more basic controls. All but one or two of the remaining controls are addressed in Chapter 14. The list box control is addressed in Chapter 22.

You can use the Control Manager to create and manage controls. An alternative method is to use the Dialog Manager to more easily create and manage controls in dialogs and alerts. In this latter case, the Dialog Manager works with the Control Manager behind the scenes. The creation and handling of controls in dialogs and alerts will be addressed in Chapter 8 and in the demonstration program associated with Chapter 14. The creation and handling areas of this chapter are limited to the use of the Control Manager to create and handle controls in document or utility windows.

Every control you create must be associated with a particular window. All the controls for a window are stored in a **control list,** a handle to which is stored in the window's window object.

Standard Controls

The term **standard controls** refers to controls whose control definition functions are provided by the system software. The term **custom controls** refers to controls that you provide yourself via a custom control definition function.

Available Standard Controls

The complete range of available standard controls is as follows.

Control	*Description*	*Variants*
Push button	A control that appears on the screen as a rounded rectangle with a title centered inside. When the user clicks a push button, the application performs the action described by the button's title. Examples include completing operations defined by a dialog and acknowledging an error message in an alert.	With title only. With color icon to left of title. With color icon to right of title.
Checkbox	A control that appears onscreen as a small square with an accompanying title. A checkbox displays one of three settings: on (indicated by a checkmark inside the box), off, or mixed (indicated by a dash inside the box).	Non-auto-toggling. Auto-toggling
Radio button	A control that appears onscreen as a small circle. A radio button displays one of three settings: on (indicated by a black dot inside the circle), off, or mixed (indicated by a dash inside the circle). A radio button is always a part of a group of related radio buttons in which only one button can be on at a time.	Non-auto-toggling. Auto-toggling

Scroll bar	A control with which the user can change the portion of a document displayed within a window. A scroll bar is a light gray rectangle with **scroll arrows** at each end. Windows can have ahorizontal scroll bar, a vertical scroll bar, or both. A vertical scroll bar lies along the right side of a window. A horizontal scroll bar runs along the bottom of a window. Inside the scroll bar is a rectangle called the **scroll box** (Mac OS 8/9) or **scroller** (Mac OS X). The rest ofthe scroll bar is called the **gray area** (Mac OS 8/9) or **track** (Mac OS X). The user can move through a document by manipulating the parts of the scroll bar.	Without live feedback. With live feedback.
Pop-up menu button	A control that is used to display a menu elsewhere than in the menu bar.	Fixed width. Variable width. Add resource. Use window font.
Bevel button	A button containing a self-descriptive icon, picture, text, or any combination of the three, that performs an action when pressed.	With small bevel (Mac OS 8/9 only). With normal bevel. With large bevel (Mac OS 8/9 only). The above with a pop-up menu either to the right or below.

Slider	A control that displays a range of values, magnitudes, or positions. A horizontally- and vertically-mobile **indicator** is used to increase or decrease the value.	Without live feedback. With live feedback. With tick marks. With directional indicator. With non-directional indicator.
Disclosure triangle	A triangular control governing how items are displayed in a list. The disclosure triangle can point right or left and down. When the disclosure triangle points to the right or left, one item is displayed in the list. When the arrow points downward, the original item and its subitems are displayed in the list.	Right-facing. Left-facing. Right-facing, auto-tracking. Left-facing, auto-tracking.
Progress bar.	A control indicating that a lengthy operation is occurring. Two types of progress bar can be used: an **indeterminate** progress bar reveals that an operation is occurring but does not indicate its duration; a **determinate** progress bar displays how much of the operation has been completed. Progress bars are also used as relevance bars on Mac OS 8/9.	(One variant only. However, the progress bar can be made determinate or non-determinate via a call to SetControlData.)
Little arrows	Up- and down-arrows accompanying a text box that contains a value, such as a date. Clicking the up arrow increases the value displayed. Clicking the down arrow decreases the value displayed.	(One variant only.)

Chasing arrows	A control that indicates through a simple animation that a background process is in progress.	(One variant only.)
Tab	A control that appears as a row of folder tabs on top of a pane. It allows multiple panes to appear in the same window.	Large, north facing. Small, south facing. Large, south facing. Small, north facing. Large, east facing. Small, east facing. Large, west facing. Small, west facing.
Separator line	A control that draws a vertical or horizontal line used to visually separate groups of controls.	(One variant only.)
Primary group box	A control that consists of a rectangular two-pixel-wide frame that may or may not contain a title. It is used to provide a well-defined area in a dialog into which text, pictures, icons or other controls can be embedded.	With text title. With checkbox title. With pop-up menu button title.
Secondary group box	A control that consists of a rectangular one-pixel-wide frame which may or may not contain a title. It is used to provide a well-defined area in a dialog into which text, pictures, icons or other controls can be embedded.	With text title. With checkbox title. With pop-up button title.

Image well	A control that is used to display non-text visual content surrounded by a rectangular frame.	(One variant only.)
Pop-up arrow	A control that simply draws the pop-up glyph.	Large, east-facing. Large, west-facing. Large, north-facing. Large, south-facing. Small, east -facing. Small, west -facing. Small, north -facing. Small, south -facing.
Placard	A rectangular control used to delineate an area in which information may be displayed.	(One variant only.)
Clock	A control that combines the features of little arrows and an edit text field into a control which displays a date and/or time.	Displays hours, minutes. Displays hours, minutes, seconds. Displays date, month, year. Displays month, year.
User pane	A general purpose control which can be used as the **root control** for a window and as an **embedder control** in which other controls may be embedded. It can also be used to hook in callback functions for drawing, hit testing, and so on.	(One variant only.)

Edit text	A control that appears as a rectangular box in which the user enters text to provide information to an application.	Normal. For passwords. For inline input.
Static text	A control that displays static (unchangeable by the user) text labels in a window.	(One variant only.)
Picture	A control used to display pictures.	Tracking. Non-tracking.
Icon	A control used to display icons.	Tracking. Non-tracking. Icon suite, tracking. Icon suite, non-tracking. All icon types, tracking. All icon types, non-tracking.
Window header	A rectangular control that is positioned along the top of a window's content region and which is used to delineate an area in which information may be displayed.	Window header. Window list view header.
List box	A control that combines a rectangular frame, scroll bar(s), and a scrolling list.	Non-autosizing. Autosizing.
Radio Group	A control that implements a radio button group.	(One variant only.)

Scrolling text box	A control that implements a scrolling text box.	Non-auto-scrolling. Auto-scrolling.
Data Browser	A control that implements a user interface component for browsing (optionally) hierarchical data structures. **Note:** This control is not addressed in this book.	
Disclosure button	A button used to hide or show specific content. Available on Mac OS X only.	(One variant only)
Edit Unicode text	Similar to the edit text controls, except that it handles Unicode text. Available on Mac OS X only.	(One variant only)
Relevance bar.	A control that indicates a level of relevance. Available on Mac OS X only.	(One variant only.)
Round button	Similar to a push button, except that it is round. Available on Mac OS X only.	Normal size. Large size.

Definition of the Term "Controls"

On Mac OS 8/9, prior to the introduction of Mac OS 8 and the Appearance Manager, a control was defined as an "on-screen object which the user can manipulate to cause an immediate action or to change settings to modify a future action." Given this previous definition, the question arises as to why such objects as, for example, separator lines and window headers are now implemented as controls. On the surface, it may appear that these objects are purely visual entities.

Part of the answer to that question has to do with the matter of Mac OS 8/9 theme-compliance introduced with the Appearance Manager. For example, using the provided separator line "control" to draw separator lines will ensure that those lines are drawn with the correct Platinum appearance (and, on Mac OS X, the correct Aqua "look") in both the activated and deactivated modes.

Another part of the answer has to do with the concept of **embedding** . The window header control, for example, is not just the visual entity it might at first appear to be; it is actually a control in which other controls may be embedded. (As will be seen, the ability to embed other controls is a powerful feature of some of the controls introduced with Mac OS 8 and the Appearance Manager.)

Since many of the new controls are not really controls "which the user can manipulate", a more accurate blanket definition might now be "any element of the user interface that is implemented by a control definition function."

Controls Addressed in This Chapter

Of the controls listed above, only those that might be termed the **basic controls** (push buttons, checkboxes, radio buttons, scroll bars, and pop-up menu buttons), together with **primary group boxes** (text title variant) and **user panes,** will be addressed in this chapter and its associated demonstration programs. These controls, with the exception of the user pane (which is invisible), are illustrated in Figs 1 and 2.

Figure 1. THE BASIC CONTROLS AND THE PRIMARY GROUP BOX (TEXT TITLE VARIANT) (MAC OS 8/9)

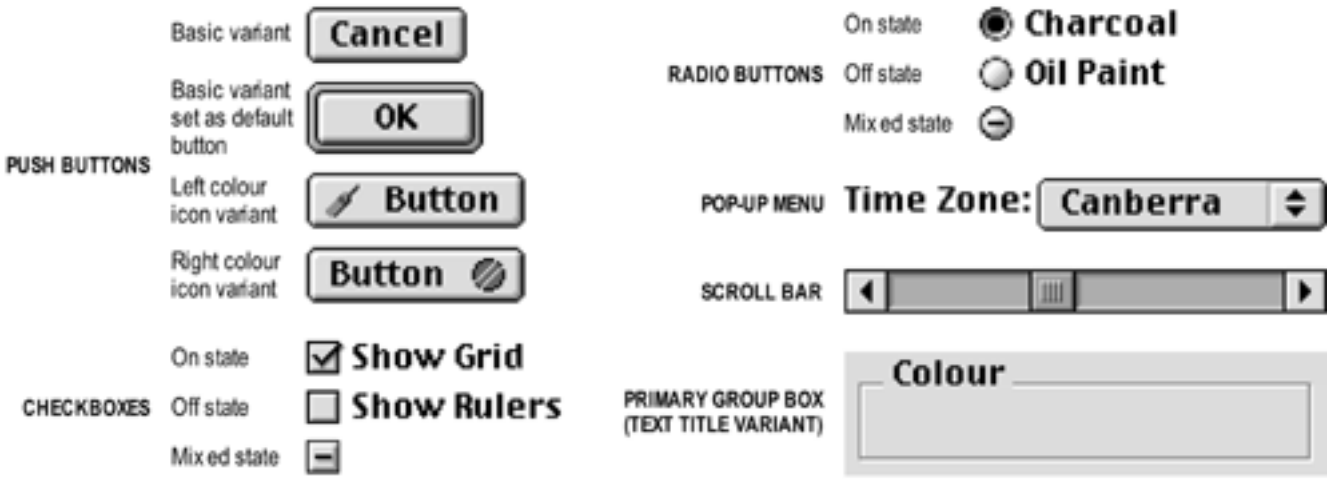
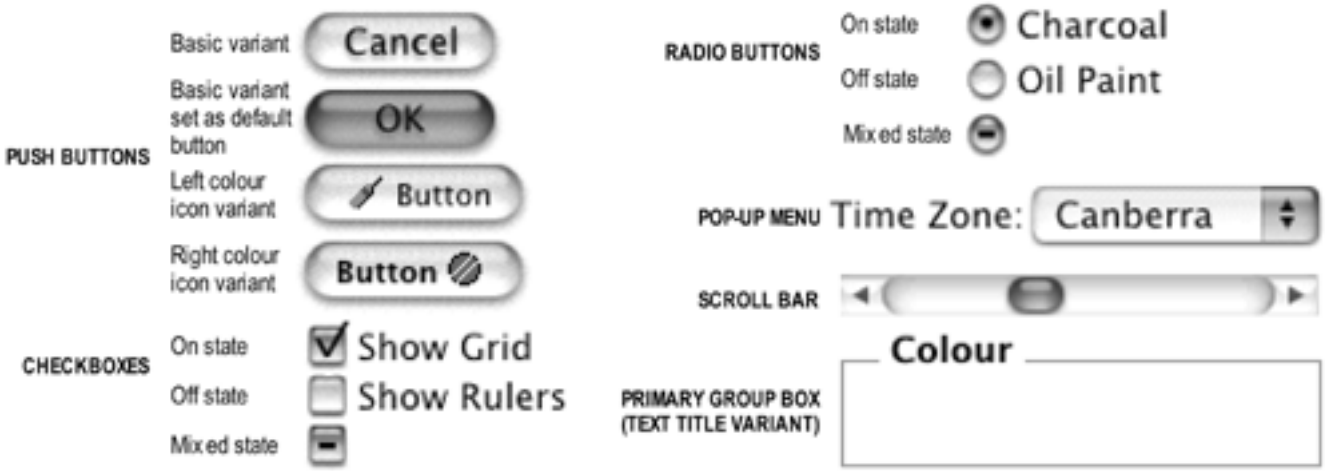


Figure 2. THE BASIC CONTROLS AND THE PRIMARY GROUP BOX (TEXT TITLE VARIANT) (MAC OS X)


PUSH BUTTONS
Basic variant
Cancel
Basic variant set as default button
OK
Left colour icon variant
Button
Right colour icon variant
Button
CHECKBOXES
On state
Show Grid
Off state
Show Rulers
Mixed state
RADIO BUTTONS
On state
Charcoal
Off state
Oil Paint
Mixed state
POP-UP MENU
Time Zone:
Canberra
SCROLL BAR
PRIMARY GROUP BOX (TEXT TITLE VARIANT)
Colour

The Control Definition Function

Control definition functions (CDEFs), which are stored as resources of type `'CDEF'`, determine the appearance and behavior of a control.

Just as a window definition function can describe variations of the same basic window, a CDEF can use a **variation code** to describe variations of the same control. You specify a particular control with a **control definition ID,** which is an integer containing the resource ID of the in the upper 12 bits and the variation code in the lower four bits.

The control definition ID is arrived at by multiplying the resource ID by 16 and adding the variation code. The following shows the control definition IDs for the standard controls and variants addressed in this chapter and its associated demonstration programs, together with the derivation of those IDs.

CDEF Resource ID	*Variation Code*	*Control Definition ID (Value)*	*Control Definition ID(Constant)*
23	0	23 * 16 + 0 = 368	kControlPushButtonProc
23	4	23 * 16 + 4 = 374	kControlPushButLeftIconProc
23	5	23 * 16 + 5 = 375	kControlPushButRightIconProc
23	1	23 * 16 + 1 = 369	kControlCheckBoxProc
23	3	23 * 16 + 3 = 371	kControlCheckBoxAutoToggleProc
23	2	23 * 16 + 2 = 370	kControlRadioButtonProc

23	4	23 * 16 + 4 = 372	kControlRadioAutoToggleButtonProc
24	0	24 * 16 + 0 = 384	kControlScrollBarProc
24	2	24 * 16 + 2 = 386	kControlScrollBarLiveProc
25	0	25 * 16 + 0 = 400	kControlPopupButtonProc
25	1	25 * 16 + 1 = 401	kControlPopupButtonProc + kControlPopupFixedWidthVariant
25	2	25 * 16 + 2 = 402	kControlPopupButtonProc + kControlPopupVariableWidthVariant
25	4	25 * 16 + 4 = 404	kControlPopupButtonProc + kControlPopupUseAddResMenuVariant
25	8	25 * 16 + 8 = 408	kControlPopupButtonProc + kControlPopupUseWFontVariant
10	0	10 * 16 + 0 = 160	kControlGroupBoxTextTitleProc
16	0	16 * 16 + 0 = 256	kControlUserPaneProc

Note that the a single CDEF caters for both horizontal and vertical scroll bars. The CDEF determines whether a scroll bar is vertical or horizontal from the rectangle you specify when you create the control.

The Basic Controls, Primary Group Boxes (Text Title Variant), and User Panes

Push Buttons

You normally use push buttons in alerts and dialogs.

In windows and dialogs containing push buttons, you should designate one push button as the **default push button,** that is, the push button the user is most likely to click. On Mac OS 8/9, the default push button is outlined. On Mac OS X it is colored and pulsing (see Fig 1).

In your application, pressing the Enter and Return keys should result in the same action as clicking on the default push button.

Checkboxes

Checkboxes are typically used in dialogs. Checkboxes provide alternative choices and act like toggle switches. Each checkbox must have a title, which should reflect two clearly opposite states.

Non-Auto-Toggling Variant

When the non-auto-toggling variant is being used, and when the user clicks a checkbox in the **off state,** your application should call `SetControlValue` to set the control to the **on state** and place a checkmark in the box (see Fig 1). When the user again clicks the checkbox, your application should call `SetControlValue` to set the control to the off state and remove the checkmark from the box.

`SetControlValue` may also be used to place a dash in the box to indicate that the control is in the **mixed state** (see Fig 1). The mixed state is a special state that indicates that a selected range of items has some in the on state and some in the off state. For example, a text formatting checkbox for bold text would be in the mixed state if a text selection contained both bold and non-bold text.

Auto-Toggling Variant

When the auto-toggling variant is being used, checkboxes automatically change between their various states (on, off, and mixed) in response to user actions. Your application need only call the function `GetControl32BitValue` to get the checkbox's new state. There

is no need to programmatically change the control's value after tracking successfully.

Radio Buttons

Like checkboxes, radio buttons are typically used in dialogs. Unlike checkboxes, radio buttons within a group are mutually exclusive.

Radio buttons are typically grouped. Each group must have a label indicating the kind of choices offered by the group, and each button must have a title indicating what it does.

Non-Auto-Toggling Variant

When the non-auto-toggling variant is being used, and when the user clicks a radio button in the off state, your application should call `SetControlValue` to

- Set that control to the **on state** and place a black dot in its circle (see Fig 1)
- Set the previously "on" button in the group to the **off state** and remove the black dot from its circle

`SetControlValue` may also be used to place a dash in the circle to indicate that the control is in the **mixed state** (see Fig 1). The mixed state is a special state that shows that a selected range has a variety of items in the on state. For example, a set of radio buttons for selecting font size might have buttons representing 10- and 12-point sizes. If a passage of text with both 10- and 12-point text was selected, both the 10 and 12 buttons would appear in the mixed state.

Auto-Toggling Variant

When the auto-toggling variant introduced with Mac OS 8.5 is being used, radio buttons automatically change between their various states (on, off, and mixed) in response to user actions. Your application need only call the function `GetControl32BitValue` to get the radio button's new state. There is no need to programmatically change the control's value after tracking successfully.

Scroll Bars

Scroll bars scroll a document within a window. Scroll bars have **scroll arrows** at each end and a movable **scroll box** (Mac OS 8/9) or **scroller** (Mac OS X). The part of a scroll bar not occupied by the scroll arrows and scroll box/scroller is called the **gray area** (Mac OS 8/9) or **track** (Mac OS X). When the user clicks in a scroll arrow or gray area/track, or drags the scroll box/scroller, your application must scroll the document as appropriate.

As previously stated, the CDEF for scroll bars supports two variants. The only difference between the two variants is that one supports **live feedback** and the other does not. In the case of scroll bars, live feedback (a generic term) may be used to perform **live scrolling** of a document in a window. Live scrolling means that, when the user attempts to drag the scroll box/scroller, the scroll box/scroller moves and the document scrolls as the user moves the mouse. (Without live scrolling, only a ghosted image of the scroll box/scroller moves. In addition, the document is only scrolled, and the scroll box/scroller proper is only redrawn at its new location, when the user releases the mouse button.)

Scroll Arrows and Gray Area/Track

When the scroll arrows are clicked, your application should move the scroll box/scroller the appropriate distance in the direction of the arrow being clicked and scroll the window contents accordingly. Each click should move the window contents one unit in the specified direction. (In a text document, one unit would typically be one line of text.)

When the gray area/track is clicked above the scroll box/scroller, your application should move the document down so that the top unit of the previous view is at the bottom of the new view, and it should move the scroll box/scroller accordingly. A similar, but upward movement, should occur when the user clicks in the gray area/track below the scroll box/scroller.

Scroll Box/Scroller

Live Feedback Variant Not Used

When the live feedback variant is not being used, and when the user drags the scroll box/scroller, the Control Manager redraws the scroll box/scroller proper in its new position, and sets the control's value accordingly, when the user releases the mouse button. You must then ascertain the position (that is, the **value**) of the scroll box/scroller and, using this value, display the appropriate portion of the document.

Live Feedback Variant Used

When the live feedback variant is being used, and when the user drags the scroll box/scroller, the Control Manager continually redraws the scroll box/scroller, and continually returns the control's position (that is, its **value**) as the scroll box/scroller moves. Once again, your application uses this value to display the appropriate portion of the document.

Proportional Scroll Boxes/Scrollers

A proportional scroll box/scroller is one whose height (vertical scroll bars) or width (horizontal scroll bars) varies in relation to the height/width of the scroll bar so as to visually represent the proportion of the document visible in the window.

Those of your application's scroll boxes/scrollers created from `'CNTL'` resources will appear as proportional scroll boxes/scrollers provided that you pass the size of the view area, in whatever units the scroll bar uses, to the function `SetControlViewSize`. The system automatically handles resizing the scroll box/scroller once your application supplies this information. (On Mac OS 8/9, the user must also have selected Smart Scrolling on in the Options tab in the Appearance control panel.) For scroll bars created programmatically using the function `CreateScrollBarControl`, you pass the size of the view area in the `viewSize` parameter.

The following functions are relevant to proportional scroll boxes/scrollers:

Function	*Description*
`GetControlViewSize`	Obtains the size of the content to which a control's size is proportioned
`SetControlViewSize`	Informs the Control Manager of the size of the content to which a control's size is proportioned

Pop-Up Menu Buttons

Pop-up menu buttons provide an alternative method of providing the user with a list of choices. The items in a pop-up menu button's menu should be mutually exclusive. Pop-up menus should be used to provide a choice of attributes, and should not be used to provide additional commands.

Primary Group Boxes (Text Title Variant)

A primary group box (text title variant) is a control that consists of a rectangular frame which may or may not contain a **title.** Group boxes are used to associate, isolate, and distinguish groups of related controls, such as a group of radio buttons.

The primary group box is an **embedder** control, meaning that you can embed other controls, such as radio buttons, checkboxes, and pop-up menu buttons, within it.

User Panes

The user pane is unique among the family of controls in that it has no visual representation. It has two main uses:

- Like the primary group box, it can be used as an embedder control-that is, other controls may be embedded within it.
- It provides a way to hook in application-defined (callback) functions, known as user pane functions, which perform actions such as drawing, hit testing, and so on.

Activating, Deactivating, Hiding, Showing, Enabling, and Disabling Controls

Activating and Deactivating Controls

A control can be either **active** or **inactive.** A control should be made inactive when it is inappropriate for your application to respond to a mouse-down event in that control. The Control Manager displays inactive controls in a dimmed state. The Control Manager displays inactive basic controls and inactive primary group boxes as shown in Figs 3 and 4.

Figure 3. THE BASIC CONTROLS AND THE PRIMARY GROUP BOX (TEXT TITLE VARIANT) IN INACTIVE MODE (MAC OS 8/9)

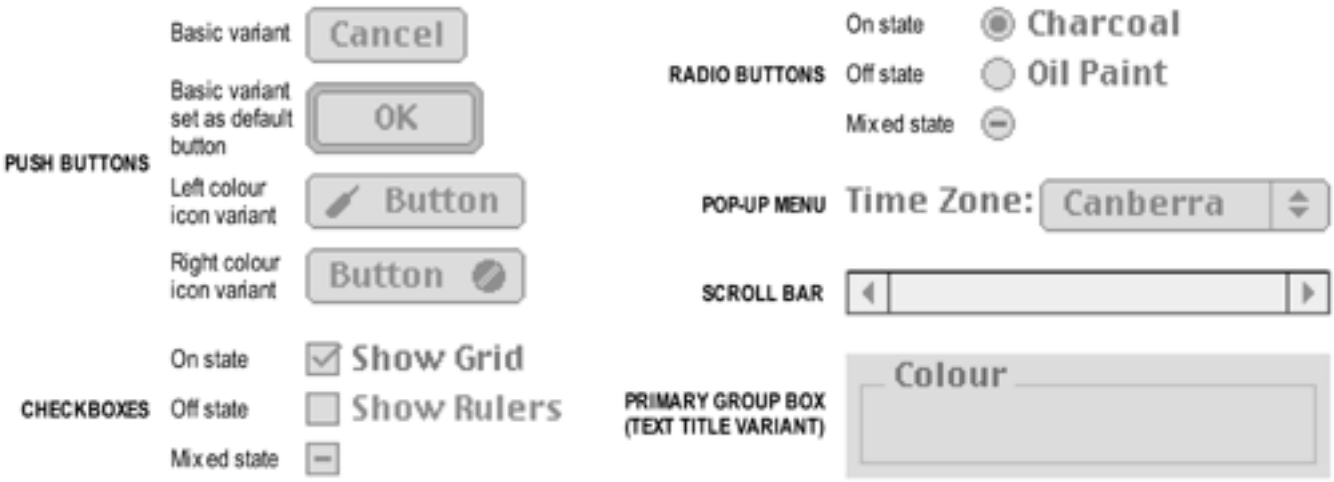
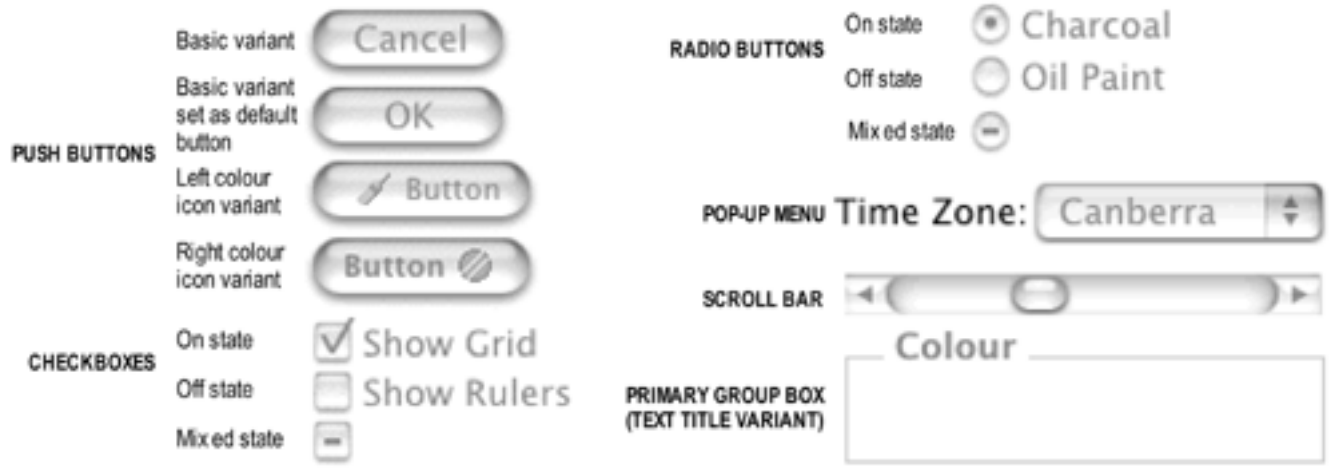


Figure 4. THE BASIC CONTROLS AND THE PRIMARY GROUP BOX (TEXT TITLE VARIANT) IN INACTIVE MODE (MAC OS X)



Activating and Deactivating Controls Other Than Scroll Bars

You use `ActivateControl` and `DeactivateControl` to make push buttons, checkboxes, radio buttons, pop-up menu buttons and primary group boxes active and inactive. You should make these controls inactive when

- They are not relevant to the current context. (But see also "Enabling and Disabling Controls.")
- The window in which they reside is not the active window.

Your application can ascertain whether a control is currently active or inactive using `IsControlActive`.

Activating and Deactivating Scroll Bars

Scroll bars become irrelevant to the current context when the document being displayed is smaller than the window in which it is being displayed. To make a scroll bar inactive in this case, you typically use `SetControlMaximum` to make the scroll bar's maximum value equal to its minimum value, which causes the Control Manager to automatically make the scroll bar inactive and display it in the inactive state. To make the scroll bar active again, `SetControlMaximum` should be used to set its maximum value larger than its minimum value.

Hiding and Showing Controls

`HideControl` may be used to hide a control. `HideControl` erases a control by filling its enclosing rectangle with the owning window's background pattern.

`ShowControl` may be used to show a control. `ShowControl` makes the control visible and immediately draws the control within its window without using your window's standard updating mechanism.

`SetControlVisibility` may be used to both hide and show a control. With regard to showing a control, this function differs from `ShowControl` in that the option is available to "show" the control without redrawing it immediately.

Your application can ascertain whether a control is currently hidden or visible using `IsControlVisible`.

Enabling and Disabling Controls

`EnableControl` and `DisableControl` may be used to enable and disable controls. `IsControlEnabled` may be used to determine whether a control is enabled or disabled.

A distinction needs to be made between a disabled control and a deactivated control. A deactivated control is simply a control in a deactivated window. (All controls in a window should be deactivated when the window is deactivated.) A disabled control, on the other hand, is a control which the user should not currently be able to manipulate. The activation state of the window is immaterial in this case.

A "delete" button, for example, should be disabled when nothing is currently selected. As another example, controls in floating windows should never be deactivated, simply because floating windows themselves should never be deactivated. However, a control in a floating window should be disabled if the user should not currently be able to manipulate it.

It follows that a control in a deactivated window must be deactivated and may be either disabled or enabled.

A problem here is that, as of the time of writing, `EnableControl`, `DisableControl`, and `IsControlEnabled` are only available on Mac OS X. Thus, on Mac OS 8/9, the only way to deny user access to a control in an active window is to deactivate it.

Visual Feedback from the Basic Controls

In response to a mouse-down event in a basic control, your application should call either `TrackControl` or `HandleControlClick`. These functions provide visual feedback when a mouse-down occurs in an active control by

- Displaying push buttons, checkboxes and radio buttons in their pressed mode
- Displaying and highlighting the items in pop-up menu buttons
- Highlighting the scroll arrows in scroll bars
- Moving the scroll box/scroller (live feedback variant of the scroll bar CDEF being used) or moving a ghosted image of the scroll box/scroller (live feedback variant not being used) when the user drags it

`HandleControlClick` was introduced with Mac OS 8 and the Appearance Manager. It is identical to `TrackControl` except that it allows modifier keys to be passed in its third parameter. Some of the newer controls, such as edit text fields and list boxes, require the ability for modifier keys to be passed in. If you use `HandleControlClick` with controls for which modifier keys are irrelevant, simply pass `0` in the `inModifiers` parameter.

Embedding Controls

As previously stated, controls (or, more usually, a group of controls) may be embedded in **embedder controls.**

The Root Control

The embedding of controls in a window requires that the window have a **root control.** The root control, which is implemented as a user pane control, is the **container** for all other window controls and is at the base of what is known as the **embedding hierarchy.**

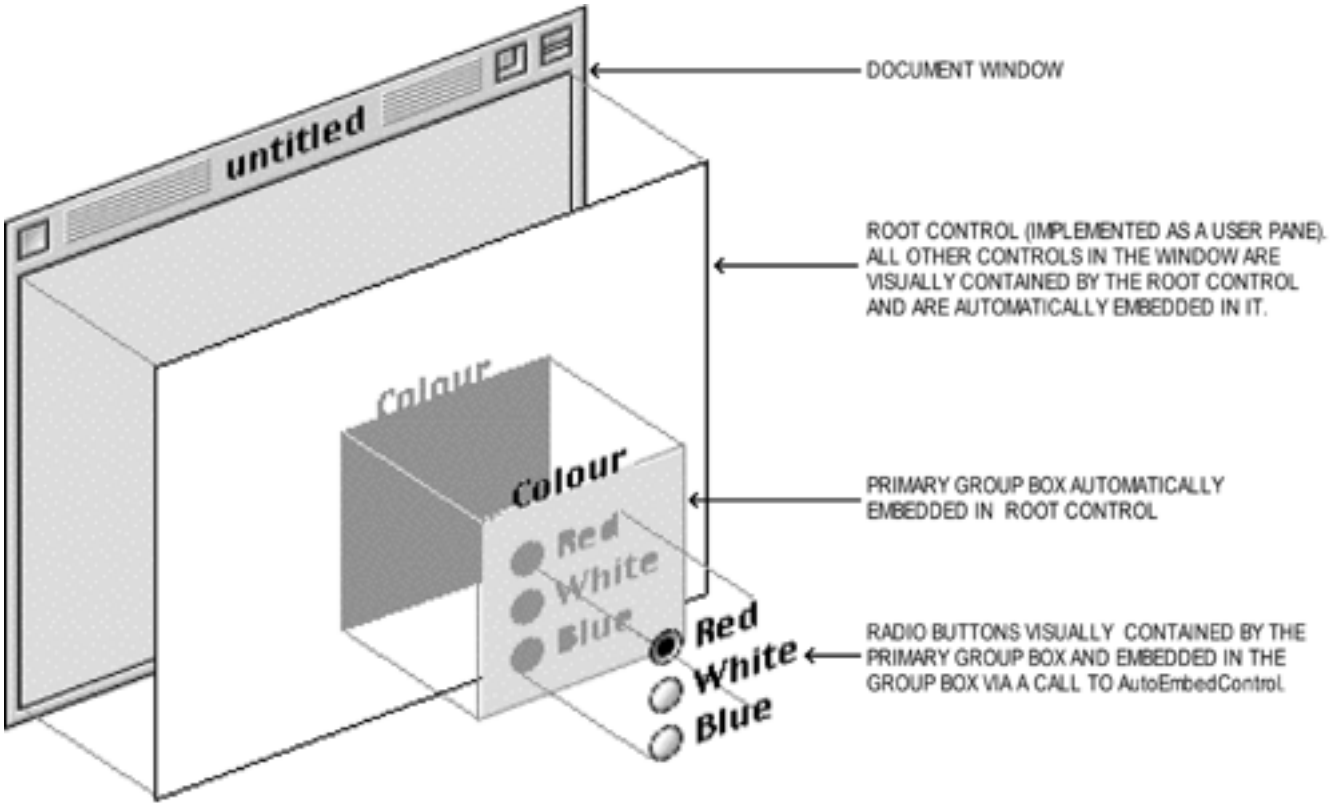
On Mac OS X, a root control is created automatically in all document windows that have at least one other control. On Mac OS 8/9, however, you must explicitly create the root control in document windows by calling `CreateRootControl`. The root control may be retrieved by calling `GetRootControl`.

Once you have created a root control, new controls will be automatically embedded in the root control when they are created. One advantage of such embedding is that, when you wish to activate and deactivate all of the window's controls on window activation and deactivation, you can do so by simply activating and deactivating the root control. (If the root control did not exist, you would have to activate and deactivate all of the window's controls individually.) You can also hide and show all of the window's control by simply hiding and showing the root control.

Other Embedders

Certain other controls also have embedding capability. One such embedder control is the primary group box. This means that you can embed, say, a group of radio buttons in a primary group box (which would, in turn, be already automatically embedded in the root control), an arrangement which is illustrated conceptually in Fig 5. By acting on the group box alone, you can then activate, deactivate, hide, show, and move all four controls as a group.

Figure 5. THE ROOT CONTROL AND EMBEDDED CONTROLS


untitled
DOCUMENT WINDOW
ROOT CONTROL (IMPLEMENTED AS A USER PANE). ALL OTHER CONTROLS IN THE WINDOW ARE VISUALLY CONTAINED BY THE ROOT CONTROL AND ARE AUTOMATICALLY EMBEDDED IN IT.
Colour
Red
White
Blue
PRIMARY GROUP BOX AUTOMATICALLY EMBEDDED IN ROOT CONTROL
RADIO BUTTONS VISUALLY CONTAINED BY THE PRIMARY GROUP BOX AND EMBEDDED IN THE GROUP BOX VIA A CALL TO AutoEmbedControl

`EmbedControl` may be used to embed a control in another (embedder) control. However, where the control to be embedded is visually contained by the embedder, as is the case with the radio buttons in Fig 5, `AutoEmbedControl` would be more appropriate.

Other Advantages of Embedding

Drawing Order

As controls are created by your application, they are added to the head of the window's control list. When those controls are drawn in the absence of an embedding hierarchy, the Control Manager starts from the top of the control list, drawing the controls in the opposite order to the order in which it they were created.

In the example in Fig 5, assume that there is no embedding hierarchy and that the radio buttons are created after the group box. This means that the group box will be drawn after the radio buttons, thus obscuring the radio buttons. An embedding hierarchy, however, enforces drawing order by drawing the embedding control before its embedded controls regardless of which is created first.

Hit Testing

Hit-testing is the process of testing whether a control is under the cursor at the time of a mouse-down event, and of identifying that control. For situations where controls are visually contained by other controls, an embedding hierarchy enforces orderly hit-testing by forcing an "inside-

out" hit test aimed at determining the most deeply nested control that is hit by the mouse.

Latency

Latency pertains to the ability of the Control Manager to remember the activation and visibility status of an embedded control when its embedder is cycled between activated and deactivated, or between visible and hidden.

For example, assume that the radio button labeled White in Fig 5 has been separately deactivated by the application. When the primary group box is deactivated, the two remaining radio buttons will also be deactivated. When the primary group box is again activated, the Control Manager remembers that the radio button labeled white was previously deactivated, and ensures that it remains in that mode.

Getting and Setting Control Data

Getting and setting control data is essentially a mechanism that allows the outside world to access a control's specialized data without exposing how that data is stored. It allows you to easily set and get control fonts, tell the push button CDEF to draw the default outline around a default push button, and many other useful things.

Each piece of information that a particular CDEF allows access to is referenced by a **tag,** which is a constant that is meaningful to the CDEF and which represents the data in question. Each tagged piece of data can be of any data type, such as a menu reference or a structure.

Control data tag constants are passed in the third parameter of the getter and setter functions `SetControlData` and `GetControlData`. The control data tag constants relevant to the basic controls are as follows:

Control Data Tag Constant	***Meaning and Data Type Returned or Set***
`kControlPushButtonDefaultTag`	Causes a push button to be drawn with the appearance of the default push button, or returns whether the push button is drawn with the default push button appearance. Data type returned or set: `Boolean`
`kControlPushButtonCancelTag`	Gets or sets whether a push button plays the Cancel button theme sound instead of the normal push button theme sound. Data type returned or set: `Boolean`. Default is `false`.

`kControlPopupButtonMenuRefTag`	Gets or sets the menu reference for a pop-up menu. Data type returned or set: `MenuRef`
`kControlPopupButtonMenuIDTag`	Gets or sets the menu ID for a pop-up menu button. Data type returned or set: `SInt16`
`kControlPopupButtonExtraHeightTag`	Gets or sets the amount of extra white space in a pop-up menu button. Data type returned or set: `SInt16`. Default is `0`.
`kControlGroupBoxTitleRectTag`	Get the rectangle that contains the title of a group box (and any associated control, such as a checkbox). Data type returned or set: `Rect`
`kControlScrollBarShowsArrowsTag`	Mac OS X only. Specifies whether the scroll arrows are to be drawn or not. Data type set: `Boolean`

The Control Object

The Control Manager stores information about individual controls in opaque data structures called **control objects.** The data type `ControlRef` is defined as a pointer to a control object:

```
typedef struct OpaqueControlRef*  ControlRef;
```

Accessor Functions

Accessor functions are provided to access the information in control objects. The accessor functions are as follows:

Function	*Description*
`GetControlOwner`	Gets a reference to the window in which the specified control is located.
`SetControlOwner`	Assigns the specified control to a specified window.
`GetControlBounds`	Sets the specified control's enclosing rectangle
`SetControlBounds`	Gets the specified control's enclosing rectangle.
`IsControlVisible`	Determines if the specified control is currently hidden or showing.
`SetControlVisibility`	Shows or hides the specified control.
`IsControlHilited`	Determines if the specified control is currently highlighted.
`HiliteControl`	Changes active/inactive status of a control or highlights a specified part of a control.
`GetControlHilite`	Determines whether the specified control is currently highlighted.
`GetControlValue`	Gets the specified control's value (16 bit) as set by `SetControlValue`.
`SetControlValue`	Sets the specified control's value (16 bit).

`GetControl32BitValue`	Sets the specified control's value as set by `SetControl32BitValue`.
`SetControl32BitValue`	Gets the specified control's value.
`GetControlMaximum`	Gets the specified control's maximum value (16 bit) as set by `SetControlMaximum`.
`SetControlMaximum`	Sets the specified control's maximum value (16 bit).
`GetControl32BitMaximum`	Gets the specified control's maximum value as set by `SetControl32BitMaximum`.
`SetControl32BitMaximum`	Sets the specified control's maximum value.
`GetControlMinimum`	Gets the specified control's minimum value (16 bit) as set by `SetControlMinimum`.
`SetControlMinimum`	Sets the specified control's minimum value (16 bit).
`GetControl32BitMinimum`	Gets the specified control's minimum value as set by `SetControl32BitMinimum`.
`SetControl32BitMinimum`	Sets the specified control's minimum value.
`GetControlDataHandle`	Gets a handle to data specific to a particular control type.
`SetControlDataHandle`	Sets a handle to data specific to a particular control type.
`GetControlAction`	Gets the universal procedure pointer to the specified control's action function.
`SetControlAction`	Sets a universal procedure pointer to the specified control's action function.
`GetControlReference`	Gets the specified control's reference constant.
`SetControlReference`	Sets the specified control's reference constant.
`GetControlTitle`	Gets the specified control's title.
`SetControlTitle`	Set the specified control's title.
`GetControlDefinition`	Gets the specified control's definition function.
`GetControlPopupMenuHandle`	Gets the specified popup menu button control's menu reference.

`SetControlPopupMenuHandle`	Sets the menu reference for a popup menu button control.
`GetControlPopupMenuID`	Gets the specified pop-up menu button control's menu ID.
`SetControlPopupMenuID`	Sets the menu ID for a pop-up menu button control.

Creating Controls

When you use the Dialog Manager to implement push buttons, radio buttons, checkboxes or pop-up menu buttons in alerts or dialogs, Dialog Manager functions automatically use Control Manager functions to create the controls for you.

For document and utility windows, you can create controls from `'CNTL'` resources or you can create them programmatically.

Creating Controls From 'CNTL' Resources

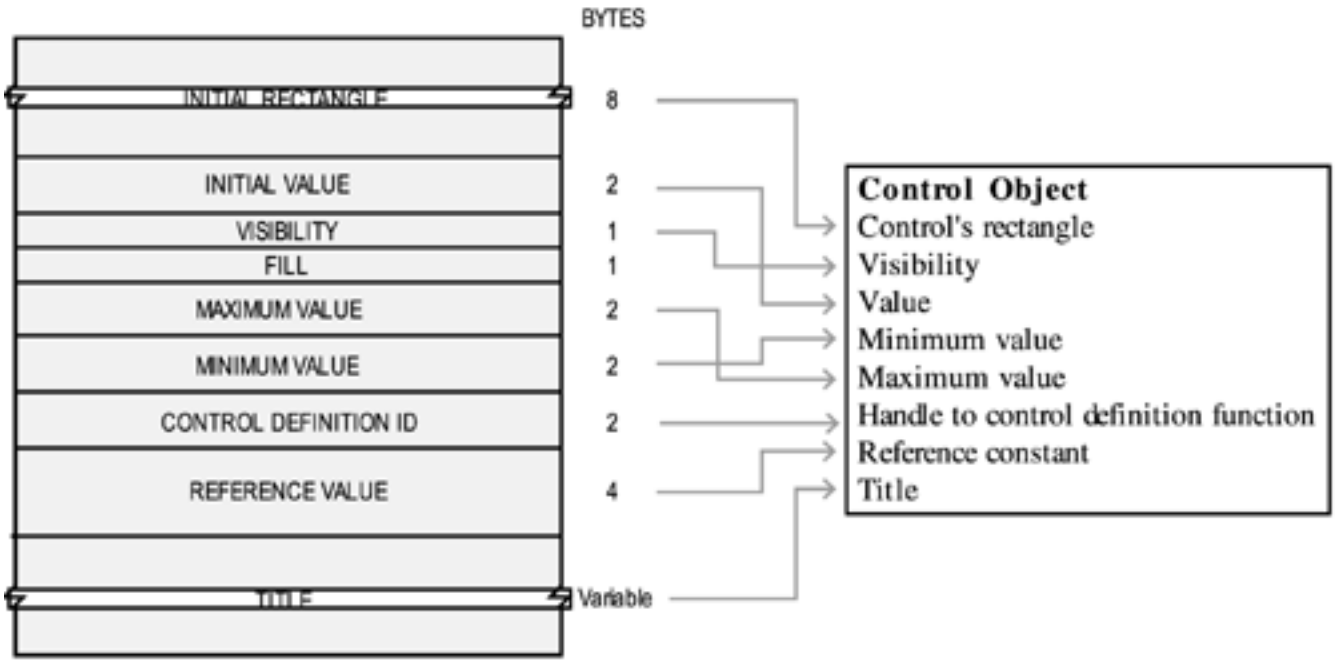
You create controls from `'CNTL'` resources using `GetNewControl`. `GetNewControl`, which takes a `'CNTL'` resource ID and a reference to the window object, creates a control object from the information in the resource, adds the control object to the control list for your window, and returns a reference to the control.

Before you can create a control using `GetNewControl`, you must, of course, first create the necessary `'CNTL'` resource. When creating resources with Resorcerer, it is advisable that you refer to a diagram and description of the structure of the resource and relate that to the various items in the Resorcerer editing windows. Accordingly, the following describes the structure of the `'CNTL'` resource.

Structure of a Compiled 'CNTL' Resource

Fig 6 shows the structure of a compiled `'CNTL'` resource and how it "feeds" the control object.

Figure 6. STRUCTURE OF A COMPILED CONTROL ('CNTL') RESOURCE



The following describes the main fields of the `'CNTL'` resource:

Field	***Description***
INITIAL RECTANGLE	A rectangle that determines the control's size and location. It is specified in local coordinates.
INITIAL VALUE	Initial value for the control. (See "Values for Controls.")
VISIBILITY	Visibility of the control. When this field contains the value `true`, `GetNewControl` draws the control immediately. When this field contains `false`, the application must use `ShowControl` when the time has come to display the control.
MAXIMUM VALUE	Maximum value of the control. (See "Values for Controls.")
MINIMUM VALUE	Minimum value for the control. (See "Values for Controls.")
CONTROL DEFINITION ID	The control definition ID. (See "The Control Definition Function.")
REFERENCE VALUE	The control's reference value, which is set up and used only by the application (except when the control is the add resource variant of the pop-up menu button, in which case this field is used to specify the resource type).
TITLE	For controls that need a title, the string for that title. For controls that do not need or do not use titles, an empty string.

Values For Controls

The following lists the initial, minimum, and maximum value settings for the basic controls and the primary group box:

Control	*Initial Value*	*Minimum Value*	*Maximum Value*
Push button	0	0	1
Checkbox	0 (initially off), or 1 (initially on), or 2 (initially in mixed state).	0	1, or 2 if mixed state checkboxes are to be used.
Radio button	0 (initially off), or 1 (initially on), or 2 (initially in mixed state).	0	1, or 2 if mixed state radio buttons are to be used.
Scroll bar	Whatever initial value is appropriate (between the minimum and maximum settings).	Whatever minimum value is appropriate. The value must be between -32768 and 32767.	Whatever maximum value is appropriate. The value must be between -32768 and 32767. When the maximum setting is equal to the minimum setting, the CDEF makes the scroll bar inactive. When the maximum setting is greater than the minimum setting, the CDEF makes the scroll bar active.

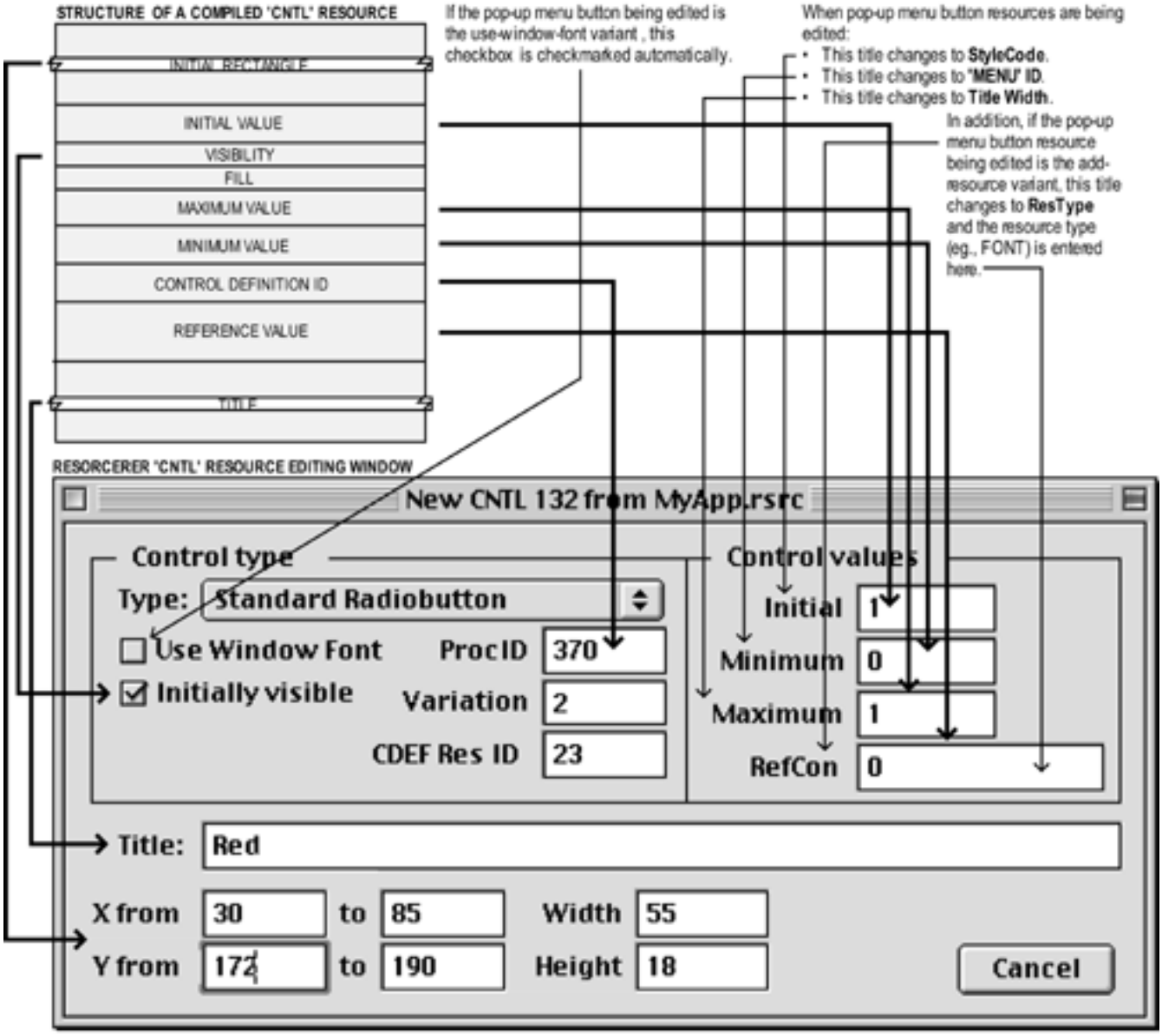
Pop-up menu button Title	A combination of values which instructs the Control Manager how to draw the control's title. (See Pop-up Menu Button Title Style Constants, below.)	Resource ID of the `'MENU'` resource.	Width (in pixels) of the title. (See Pop-up Menu Button Width, below.)
Primary group	Ignored if the group box is the text title variant.	Ignored if the group box is the text title variant.	Ignored if the group box box is the text title variant.

Note that the title of each of the three value fields is somewhat of a misnomer in the case of the pop-up menu button. These fields are thus said to be "overloaded."

Creating 'CNTL' Resources Using Resorcerer

Fig 7 shows a `'CNTL'` resource being created with Resorcerer.

Figure 7. CREATING A 'CNTL' RESOURCE USING RESORCERER


STRUCTURE OF A COMPILED 'CNTL' RESOURCE
INITIAL RECTANGLE
INITIAL VALUE
VISIBILITY
FILL
MAXIMUM VALUE
MINIMUM VALUE
CONTROL DEFINITION ID
REFERENCE VALUE
TITLE
If the pop-up menu button being edited is the use-window-font variant, this checkbox is checkmarked automatically.
When pop-up menu button resources are being edited:
• This title changes to StyleCode.
• This title changes to 'MENU' ID.
• This title changes to Title Width.
In addition, if the pop-up menu button resource being edited is the add-resource variant, this title changes to ResType and the resource type (eg., FONT) is entered here.
RESORCERER 'CNTL' RESOURCE EDITING WINDOW
New CNTL 132 from MyApp.rsrc
Control type
Type: Standard Radiobutton
Use Window Font
Initially visible
ProcID 370
Variation 2
CDEF Res ID 23
Control values
Initial 1
Minimum 0
Maximum 1
RefCon 0
Title: Red
X from 30 to 85
Y from 172 to 190
Width 55
Height 18
Cancel

Creating Controls Programmatically

You can also create controls programmatically. The following functions, which were introduced with Carbon, may be used to programmatically create the controls described in this chapter:

Function	*Parameters*
`CreatePushButtonControl`	Rectangle, title
`CreatePushButtonWithIconControl`	Rectangle, title, address of `ControlButtonContentInfo` structure, icon alignment
`CreateRadioButtonControl`	Rectangle, title, auto-toggle/non-autotoggle
`CreateCheckBoxControl`	Rectangle, title, autotoggle/non-autotoggle

`CreateScrollBarControl`	Rectangle, control value, minimum value, maximum value, size of view area, live-feedback/ non-live-feedback, UPP to control action function
`CreatePopupButtonControl`	Rectangle, title, menu ID, variable-width/non-variable-width, title width, title justification, title style.
`CreateGroupBoxControl`	Rectangle, title, primary/secondary
`CreateUserPaneControl`	Rectangle, features

Push Button Content and Icon Alignment

One of the parameters in calls to `CreatePushButtonWithIconControl` is the address of a structure of type `ControlButtonContentInfo`:

```
struct ControlButtonContentInfo
{
  ControlContentType contentType;
  union
  {
    SInt16      resID;
    CIconHandle cIconHandle;
    Handle      iconSuite;
    IconRef     iconRef;
    PicHandle   picture;
    Handle      ICONHandle;
  }  u;
} ;
typedef struct ControlButtonContentInfo ControlButtonContentInfo;
```

If, for example, you wish to specify a color icon for the button's icon content, you would assign `kControlContentCIconRes` to the `contentType` field and the icon's resource ID to the `resID` field.

Another parameter in calls to `CreatePushButtonWithIconControl` is a value of type `ControlPushButtonIconAlignment`. Relevant constants are

```
kControlPushButtonIconOnLeft
kControlPushButtonIconOnRight
```

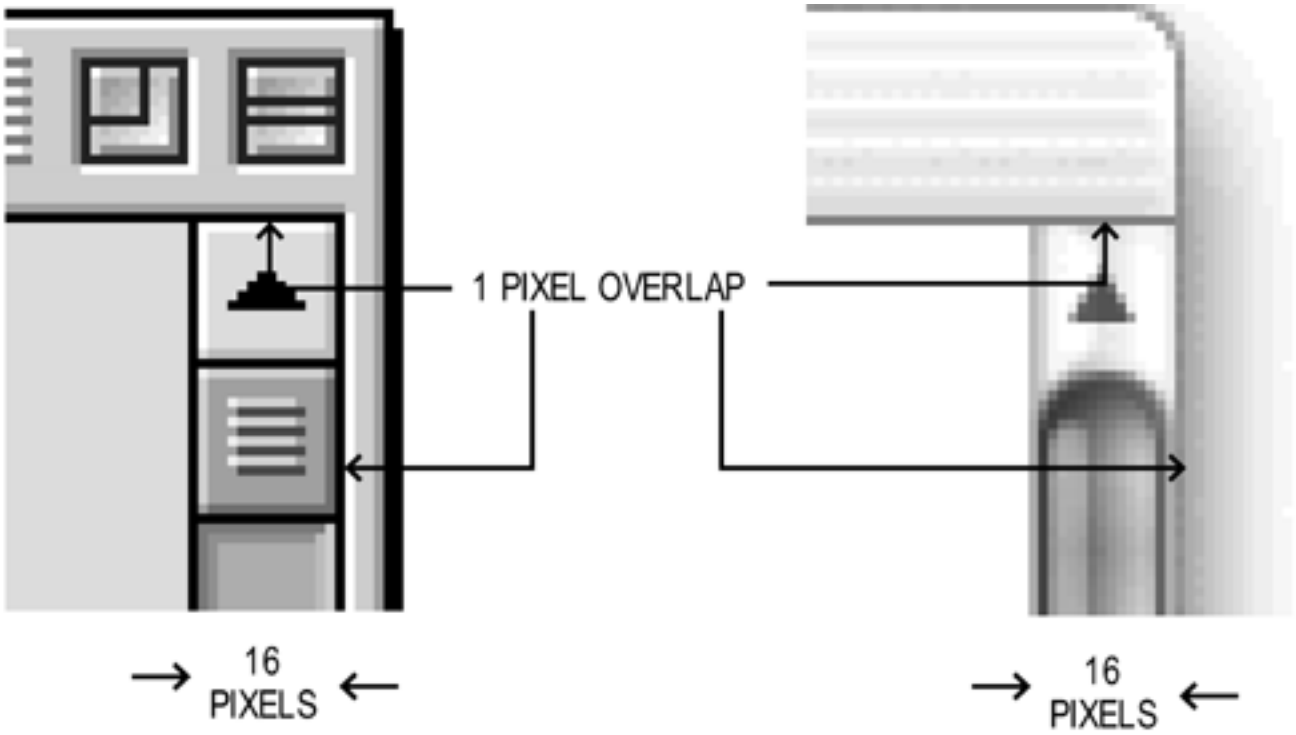
Additional Considerations-Scroll Bars

When creating scroll bars, you typically call `GetNewControl` or `CreateScrollBarControl` immediately after you create the window and then use `MoveControl`, `SizeControl`, `SetControlMaximum` and `SetControlValue` to adjust the size, location and value settings. (For example, for a window displaying a text document, you would typically calculate the number of lines of text and set the vertical scroll bar's maximum value according to the line count and the window's current height. You would set the control's value according to the part of the document to be initially displayed.)

Most applications allow the user to change the size of windows, add information to the document and remove information from the document. It is therefore necessary, in your window handling code, to calculate a changing maximum setting based on the document's current size and its window's current size. For new documents which have no content to scroll, assign an initial value of 0 as the maximum setting (which will, as previously stated, make the scroll bars inactive). Thereafter, your window-handling code should set and maintain the maximum setting.

A scroll bar for a document window is, by convention, 16 pixels wide (vertical scroll bars) and 16 pixels high (horizontal scroll bars). The Control Manager draws one-pixel lines around the scroll bar, based on the rectangle enclosing the scroll bar. As shown in Fig 8, these outside lines should overlap the inside lines of the window frame.

Figure 8. CORRECT OVERLAP OF SCROLL BAR ON WINDOW FRAME



The following calculations determine the rectangle for a vertical scroll bar for a document window:

Note

Do not include the title bar area in these calculations.

Coordinate	*Calculation*
Top	Combined height of any items above the scroll bar-1.
Left	Width of window-15.
Bottom	Height of window-14.
Right	Width of window + 1.

The following calculations determine the rectangle for a horizontal scroll bar for a document window.

Coordinate	*Calculation*
Top	Height of window-15.
Left	Combined width of any items to the left of the scroll bar-1.
Bottom	Height of window + 1.
Right	Width of window-14.

When the user resizes the window, the initial maximum settings and location, as specified in the `'CNTL'` resource or `CreateScrollBarControl` call, must therefore be changed dynamically by the application as required. Typically, this is achieved by storing handles to each scroll bar in a document structure associated with the window and then using Control Manager functions to change control settings.

Additional Considerations-Pop-up Menu Buttons

Pop-up Menu Button Title Style Constants

The constants and values for the initial value for pop-up menu buttons are as follows:

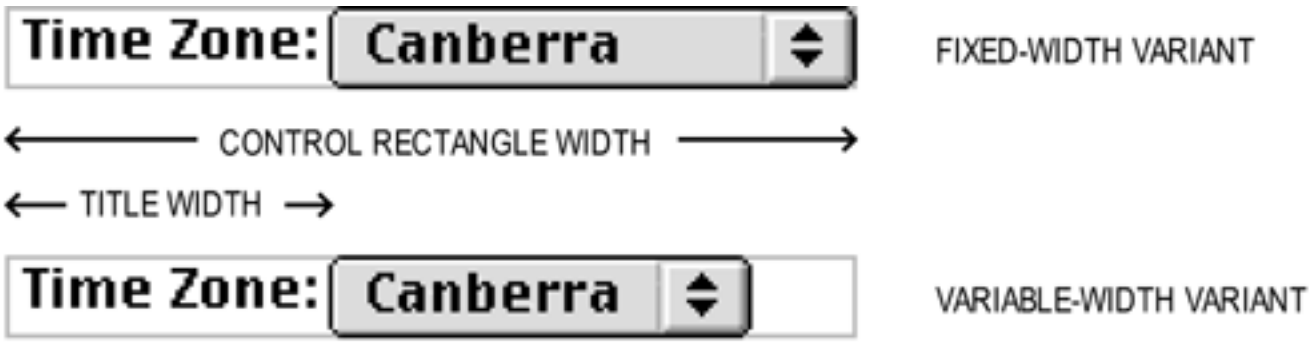
Constant	*Value*	*Meaning*
`popupTitleBold`	`0x0100`	Boldface font style
`popupTitleItalic`	`0x0200`	Italic font style
`popupTitleUnderline`	`0x0400`	Underline font style

`popupTitleOutline`	`0x0800`	Outline font style
`popupTitleShadow`	`0x1000`	Shadow font style
`popupTitleCondense`	`0x2000`	Condensed text
`popupTitleExtend`	`0x4000`	Extended text

Pop-up Menu Button Title Width

Fig 9 shows the relationship between the title width and the enclosing rectangle of a pop-up menu box.

Figure 9. POP-UP MENU BUTTON TITLE WIDTH AND CONTROL RECTANGLE WIDTH



Menu Width Adjustment

If the base variant or the variable width variant is being used, and whenever the pop-up menu button is redrawn, the CDEF calls `CalcMenuSize` to calculate the size of the menu associated with the control. If the sum of the width of the title, the longest item in the menu, the arrows, and a small amount of "white space" is less than the width of the control rectangle, the width of the pop-up button will be reduced for drawing purposes (see Fig 9). If the calculated width is greater than the width of the control rectangle, the longer menu items will be truncated with an added ellipsis so that the drawn pop-up will not exceed the width of the control rectangle.

Menu Items and Control Values

When the control is created, the first menu item and the number of items in the pop-up menu button are stored in the control object. When the user chooses a different menu item, the Control Manager changes the item number value stored in the control object to that item number.

Adding Resource Names As Items

If the add resource variant of the pop-up menu button is being used, the control reference

constant value in the control object is coerced to type `ResType` and `AppendResMenu` is called to add items of that type. For example, if you specify `FONT` in the `ResType` item in the Resorcerer `'CNTL'` resource editing window, the CDEF appends a list of fonts installed in the system to the menu specified at the `'MENU' ID` item. In this situation, you can still store a reference constant in the control object by calling `SetControlReference` after the control has been created.

Setting the Font of a Control's Title

You can set the font of any control's title independently of the system font or window font. To set the font of a control's title, you pass a pointer to a **control font style structure** in the `inStyle` parameter of the function `SetControlFontStyle`.

Control Font Style Structure

```
struct ControlFontStyleRec
{
  SInt16   flags;
  SInt16   font;
  SInt16   size;
  SInt16   style;
  SInt16   mode;
  SInt16   just;
  RGBColor foreColor;
  RGBColor backColor;
} ;
typedef struct ControlFontStyleRec ControlFontStyleRec;
typedef ControlFontStyleRec *ControlFontStylePtr;
```

Field Descriptions

`flags` Specifies which fields of the structure should be applied to the control (see "Control Font Style Flag Constants"). If no flags are set, the control uses the system font (or, for control variants which use the window font, the window font).

`font` Specifies the ID of the font family to use. Alternatively, a meta font constant may be assigned to this field (see "Meta Font Constants").

`size` If the `kControlUseSizeMask` is set in the `flags` field, specifies the point size of the text. If the `kControlAddFontSizeMask` bit is set in the `flags` field, specifies the size to *add* to the current point size of the text. Alternatively, a meta font constant may be assigned to this field (see "Meta Font Constants").

`style` Specifies the style to apply to the text. The bit numbers and the styles they represent are

as follows:

Bit Number	*Style*
`0`	Bold
`1`	Italic
`2`	Underline
`3`	Outline
`4`	Shadow
`5`	Condensed
`6`	Extended

If all bits are clear, the plain font style is specified.

`mode` Specifies how characters are drawn in the bit image. (For a discussion of transfer modes, see Chapter 12.)

`just` Specifies text justification. The relevant constants are as follows:

Constant	*Value*
`teFlushDefault`	0
`teCenter`	1
`teFlushRight`	2
`teFlushLeft`	3

`foreColor` Specifies the RGB (red-green-blue) color to use when drawing the text.

`backColor` Specifies the RGB color to use when drawing the background behind the text. (Note that, in certain text modes, background color is ignored.)

Control Font Style Flag Constants

You can pass one or more of the following control font style flag constants in the `flags` field of the control font style structure to specify those fields of the structure that are be applied to the control. If none of the flags of the `flags` field are set, the control uses the system font unless a control with a variant that uses the window font has been specified.

Constant	*Value*	*Meaning*
`kControlUseFontMask`	`0x0001`	The `font` field of the control font style structure is applied to the control.
`kControlUseFaceMask`	`0x0002`	The `style` field of the control font style structure is applied to the control. Ignored if you specify a meta font value (see "Meta Font Constants").
`kControlUseSizeMask`	`0x0004`	The `size` field of the control font style structure is applied to the control. Ignored if you specify a meta font value (see "Meta Font Constants").
`kControlUseForeColorMask`	`0x0008`	The `foreColor` field of the control font style structure is applied to the control. Applies only to static text controls.
`kControlUseBackColorMask`	`0x0010`	The `backColor` field of the control font style structure is applied to the control. Applies only to static text controls.
`kControlUseModeMask`	`0x0020`	The text mode specified in the `mode` field of the control font style structure is applied to the control.
`kControlUseJustMask`	`0x0040`	The `just` field of the control font style structure is applied to the control.
`kControlUseAllMask`	`0x00FF`	All flags in this mask will be set except `kControlUseAddFontSizeMask`.

`kControlUseAddFontSizeMask`	`0x0100`	The Dialog Manager will add a specified font size to the `size` field of the control font style structure. Ignored if you specify a meta font value (see "Meta Font Constants").

Meta Font Constants

You can use the following meta font constants in the `font` field of the control font style structure to specify the style, size, and font family of a control's font. You should use these meta font constants whenever possible because the system font can change, depending upon the current theme. If none of these constants are specified, the control uses the system font unless a control with a variant that uses the window font has been specified.

Constant	*Value*	*Meaning In Roman Script System*
`kControlFontBigSystemFont`	`-1`	Use the system font
`kControlFontSmallSystemFont`	`-2`	Use the small system font
`kControlFontSmallBoldSystemFont`	`-3`	Use the small emphasized system font
`kControlFontViewSystemFont`	`-4`	Use the views font

Another advantage of using these meta font constants is that you can be sure of getting the correct font on a Macintosh using a different script system, such as kanji.

Setting and Getting Control IDs

The following functions, which were introduced with Carbon, may be used to set and get a control's ID, and to get a reference to a control using the control's ID:

Function	*Description*
`SetControlID`	Set a control's ID
`GetControlID`	Get a control's ID
`GetControlByID`	Get a reference to a control using the control's ID

Updating, Moving, and Removing Controls

Updating Controls

When your application receives an update event for a window containing controls, it should call `UpdateControls` between the `BeginUpdate` and `EndUpdate` calls in its updating function.

Note that when you use the Dialog Manager to implement push buttons, radio buttons, checkboxes or pop-up menu buttons in alerts or dialogs, Dialog Manager functions automatically use Control Manager functions to update the controls for you.

Moving Controls

You can change the position of a control using `MoveControl`, which erases the control, offsets the control's control rectangle, and redraws it at the specified new location

Removing Controls

`DisposeControl` may be used to remove a control from a window, delete it from the window's control list, and release the control object and associated data structures from memory. `KillControls` will dispose of all of a window's controls at once.

Handling Mouse Events in Controls

Overview

For mouse events in controls, you usually perform the following tasks:

- Use `FindWindow` to determine the window in which the mouse-down event occurred.
- If the mouse-down event occurred in the content region of the active window, use `FindControl` to determine whether the event occurred in a control and, if so, which control.
- Call `TrackControl` or `HandleControlClick` to handle user interaction with the control as long as the user holds the mouse button down. The `actionProc` parameter passed to `TrackControl`, or the `inAction` parameter passed to `HandleControlClick`, should be as follows:
    - `NULL` for push buttons, checkboxes and radio buttons.
    - For scroll arrows and gray areas/tracks of scroll bars, a universal procedure pointer which invokes an application-defined **action function** which, in turn, causes the document to scroll as long as the user holds the mouse button down.
    - For the scroll box/scroller of scroll bars:
        - `NULL` if the non-live-feedback variant is being used.
        - If the live-feedback variant is being used, a pointer which invokes an application-defined action function which, in turn, causes the document to scroll while the scroll box/scroller is being dragged.
    - `(ControlActionUPP) -1` for pop-up menu buttons. This causes `TrackControl` and `HandleControlClick` to use the action function defined within the pop-up CDEF, a pointer to which is stored in the control object.

  Note that, as an alternative to passing universal procedure pointers in the `actionProc` parameter of `TrackControl`, or the `inAction` parameter of `HandleControlClick`, you can preset the action function by passing the universal procedure pointer in the `actionProc` parameter of `SetControlAction`. (Ordinarily, you would call `SetControlAction`

immediately after the control is created.) In this case, you must pass `(ControlActionUPP) -1` in the `actionProc` and `inAction` parameters of `TrackControl` and `HandleControlClick`.

- When `TrackControl` or `HandleControlClick` reports that the user has released the mouse button with the cursor in a control, respond appropriately-that is
    - Perform the task identified by the push button title if the cursor is over a push button.
    - Toggle the value of the checkbox when the cursor is over a checkbox. (The Control Manager then redraws or removes the checkmark, as appropriate.)
    - Turn on the radio button, and turn off all other radio buttons in the group, when the cursor is over an active radio button.
    - Show more of the document in the direction of the scroll arrow when the cursor is over the scroll arrow or gray area/track of a scroll bar, and move the scroll box/scroller accordingly.
    - If the non-live-feedback scroll bar variant is being used, and when the cursor is over the scroll box/scroller, determine where the user has dragged the scroll box/scroller, and then display the corresponding portion of the document.
    - Use the new setting chosen by the user when the cursor is over a pop-up menu button.

Determining a Mouse-Down Event in a Control

`FindControl` will return both a reference to the control as well as a **control part code** when a mouse-down event occurs in a visible, active control. `FindControl` will set the control reference to `NULL` and return 0 as the control part code when a mouse-down occurs in an invisible or inactive control, or when the cursor is not in a control.

A control part code is an integer from 1 to 255. Part codes are assigned to a control by its CDEF. The CDEFs for the basic controls define the following part codes:

Constant	*Value*	*Meaning*

`kControlNoPart`	`0`	Event did not occur in any control. Also unhighlights any highlighted part of the control when passed to the `HiliteControl` function.
`kControlLabelPart`	`1`	Event occurred in the label of a pop-up menu button.
`kControlMenuPart`	`2`	Event occurred in the menu of a pop-up menu button.
`kControlButtonPart`	`10`	Event occurred in a push button.
`kControlCheckBoxPart`	`11`	Event occurred in a checkbox.
`kControlRadioButtonPart`	`12`	Event occurred in a radio button.
`kControlUpButtonPart`	`20`	Event occurred in the up (or left) scroll arrow of a scroll bar.
`kControlDownButtonPart`	`21`	Event occurred in the down (or right) button of a scroll bar.
`kControlPageUpPart`	`22`	Event occurred in the page-up part of a scroll bar.
`kControlPageDownPart`	`23`	Event occurred in the page-down part of a scroll bar.
`kControlIndicatorPart`	`129`	Event occurred in the scroll box/scroller of a scroll bar.

A newer function (`FindControlUnderMouse`) will return a reference to the control even if no part was hit and can determine whether a mouse-down event has occurred even if the control is deactivated, whereas `FindControl` will not.

Tracking the Cursor in a Control

When the call to `FindControl` determines that the cursor was in a control when the user pressed the mouse button, you should call `TrackControl` or `HandleControlClick` to follow and respond to the user's movements.

You can use an action function to perform additional actions while the user holds the mouse button down. Typically, action functions are used to continuously scroll the window's contents while the cursor is on a scroll arrow or gray area/track of a scroll bar, or in the scroll box/scroller of a live-feedback scroll bar. (As previously stated, you pass a

pointer to this action function in the `actionProc` parameter of `TrackControl` or the `inAction` parameter of `HandleControlClick`).

`TrackControl` and `HandleControlClick` return the control's control part code if the user releases the mouse button while the cursor is still inside the control part, or `kControlNoPart` (`0`) if the cursor is outside the control part when the button is released. Your application should then respond appropriately.

Determining and Changing Control Settings, and Getting a Control's Kind

Determining and Changing Control Settings

Your application often needs to determine the current setting and other values of a control when the user clicks the control. When the user clicks a non-auto-toggling checkbox, for example, your application must determine whether the box is currently checked before deciding whether to clear or draw the checkmark.

Your application can use the control value accessor functions described at The Control Object, above, to get and set a control's value, minimum value, and maximum value.

Getting a Control's Kind

You can determine a control's kind using `GetControlKind`. The control's kind is returned in a structure of type `ControlKind`:

```
struct ControlKind
{
  OSType signature;  // kControlKindSignatureApple ('appl') for all system controls
  OSType kind;
} ;
typedef struct ControlKind ControlKind;
```

For the controls addressed in this chapter, the following constants pertain to the `kind` field:

Constant	*Value*	*Control Kind*
`kControlKindPushButton`	`'push'`	Push button
`kControlKindPushIconButton`	`'picn'`	Push button (color icon variant)
`kControlKindCheckBox`	`'cbox'`	Checkbox
`kControlKindRadioButton`	`'rdio'`	Radio button
`kControlKindScrollBar`	`'sbar'`	Scroll bar
`kControlKindPopupButton`	`'popb'`	Pop-up menu button
`kControlKindGroupBox`	`'grpb'`	Primary group box (text title variant)
`kControlKindUserPane`	`'upan'`	User pane

An alternative method is to call `GetControlData` with `kControlKindTag` passed in the `inTagName` parameter and the address of a variable of type `ControlKind` passed in the `inBuffer` parameter.

A problem here is that, as of the time of writing, both `GetControlKind` and the alternative method were only available on Mac OS X.

Moving and Resizing Scroll Bars

When the user resizes your windows, your application must resize and move that window's scroll bars. The steps involved are

- Resize the window.
- Make each scroll bar invisible using `HideControl`.
- Move the scroll bars to the appropriate edges of the window using `MoveControl`.
- Lengthen or shorten each scroll bar (as appropriate) using `SizeControl`.
- After recalculating the maximum settings, update the settings and redraw the scroll boxes/scrollers using `SetControlMaximum` and `SetControlValue`.
- Make each scroll bar visible at its new location using `ShowControl`.

Each of the functions involved require a reference to the relevant scroll bar control. When your application creates a window, it should store references for each scroll bar in a document structure associated with that window.

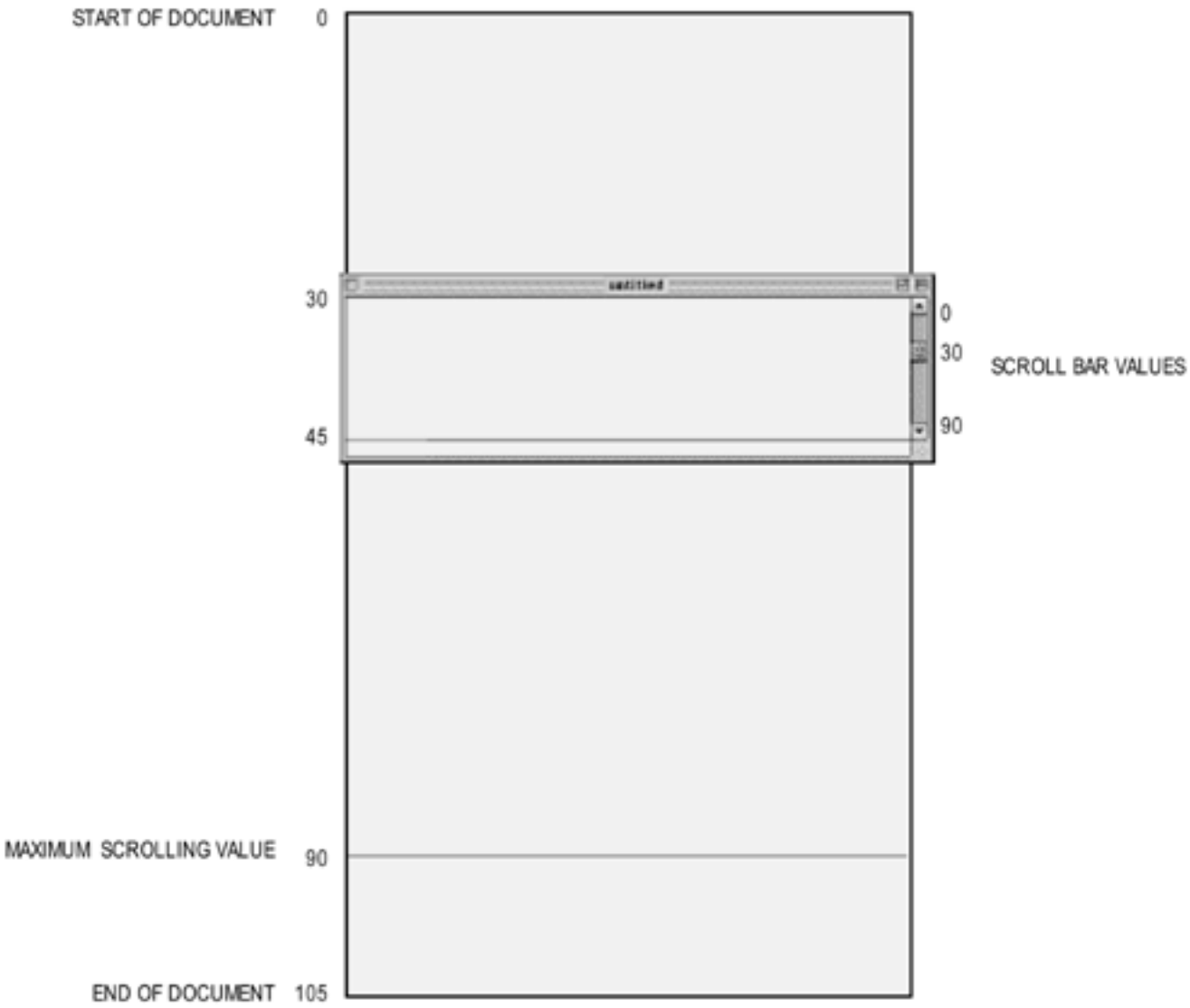
Scrolling Operations With Scroll Bars

Scrolling Basics

Relationship Between Document, Window, and Scroll Bar

The relationship between a document and a window, and their representation in a scroll bar, is shown in Fig 10.

Figure 10. RELATIONSHIP BETWEEN THE DOCUMENT, THE WINDOW, AND THE SCROLL BAR



Distance and Direction to Scroll

When the user scrolls a document using scroll bars, your application must first determine the distance and direction to scroll. The distance to scroll is as follows:

- When the user drags the scroll box/scroller to a new location, your application should scroll the document a corresponding distance.
- For a click on a scroll arrow, you should scroll by a distance appropriate to your application. For example, a text editing application might scroll one line of text for one click in the vertical scroll bar.
- For a click in the gray area/track, you should scroll by a distance appropriate to your application. For example, a text editing application should ordinarily scroll by a distance equal to the height of the window less one text line.

Direction to scroll is determined by whether the scrolling distance is expressed as a positive or negative number. The scrolling distance will be expressed as a negative number if the user scrolls down from the beginning of the document, and as a positive number if the user scrolls up towards the beginning of the document.

Scrolling the Pixels

With the distance and direction to scroll determined, the next step is to scroll the pixels displayed in the window by that distance and in that direction. Typically, `ScrollRect` is used for that purpose.

Moving the Scroll Box/Scroller

If the user did not affect the scroll using the scroll box/scroller, the scroll box/scroller must then be repositioned using `SetControlValue`.

Updating the Window

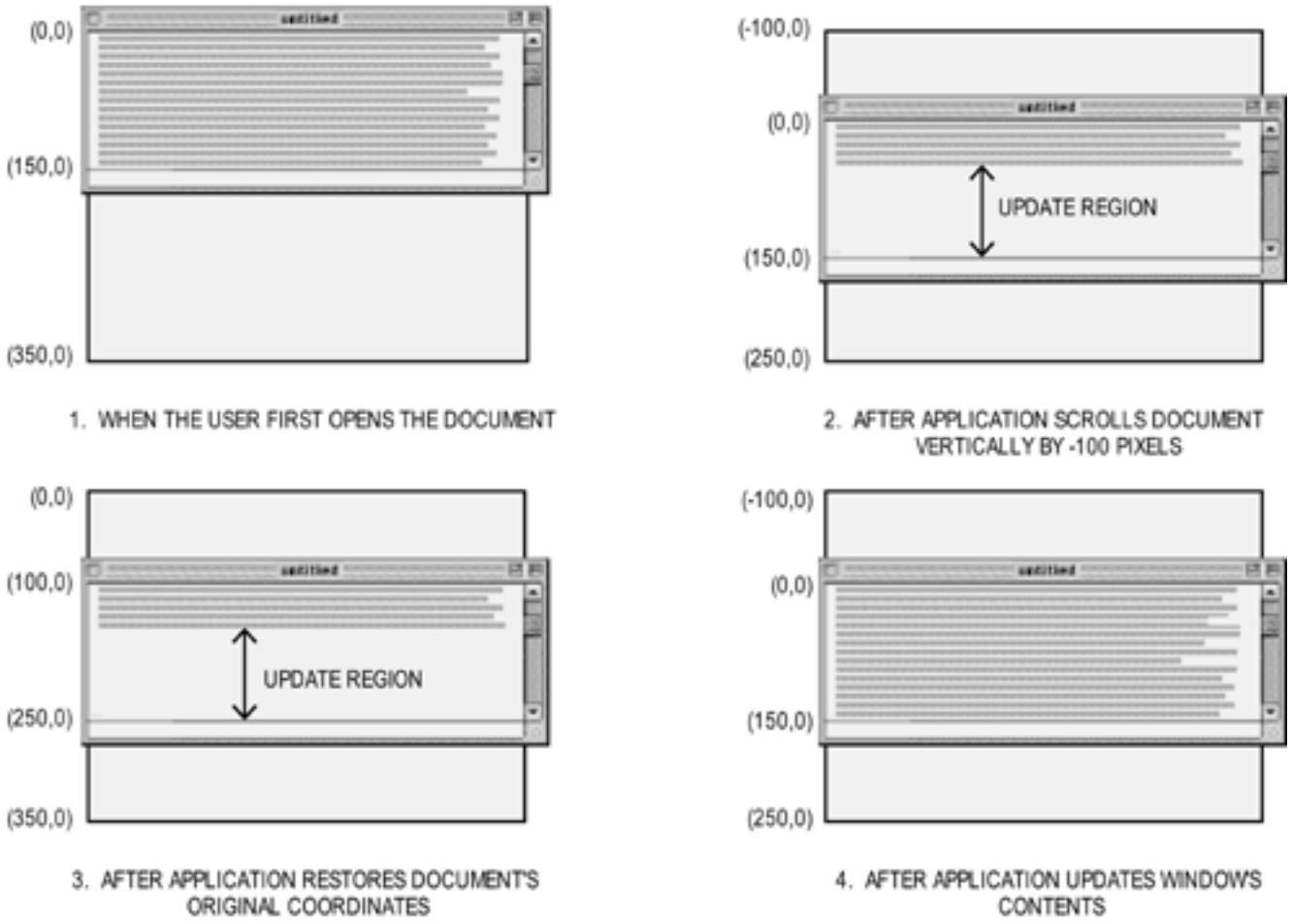
The final step is to either call a function which generates an update event or directly call your application's update function. Your application's update function should call `UpdateControls` (to update the scroll bars) and redraw the appropriate part of the document in the window.

Scrolling Example

Half the complexity of scrolling lays in ensuring that that part of the document which is displayed in the window correlates with the scroll bar control value, and vice versa, at all times.

Consider the left-top of Fig 11. This newly opened document consists of 35 lines of text with a consistent line height of 10 pixels. The document is, therefore, 350 pixels long. The upper-left point of the document is identical to the window origin-that is, both are at (0,0).

Figure 11. SCROLLING A DOCUMENT IN A WINDOW



The window height is 150 pixels (15 lines of text). Thus the maximum setting for the scroll bar is equivalent to 200 pixels down in the document. (As shown in Fig 10, a vertical scroll bar's maximum setting equates to the length of the document minus the height of the window.)

Now assume that the user drags the scroll box/scroller *down* the vertical scroll bar. This means that your application must move the text *up* by the appropriate amount. Moving a document up means a negative scrolling value.

Suppose the application, using `GetControlValue`, determines that the scroll bar's control value is 100. The application must then call `ScrollRect` to shift the bits displayed in the window by a distance of -100 pixels (that is, 10 lines of text). As shown at the top-right of Fig 11, five lines from the bottom of the previous window display now appear at the top of the window.

Note that, in all this, `ScrollRect` does not change the coordinate system of the window; it simply moves the bits in the window to new coordinates that are still in the window's local coordinate system. In terms of the window's local coordinate system, then, the upper left corner of the document is now at (-100,0).

To facilitate updating of the window (see the update region in Fig 11), `SetOrigin` must now be used to change the local coordinate system of the window so that the application can treat the upper left corner of the document as again lying at (0,0). This restoration of the document's original coordinate space makes it easier for the application to determine which lines of the document to draw in the update region of the window (see bottom-left of Fig 11).

The application now updates the window by drawing lines 16 to 24, which it stores in its document structure as beginning at (160,0) and ending at (250,0).

Finally, because the Window and Control Managers always assume that the window's upper-left point is at (0,0) when they draw in the window, the window origin cannot be left at (100,0). Accordingly, the application must use `SetOrigin` to reset it to (0,0) after performing its own drawing, (See bottom-right of Fig 11.)

To summarize:

- The user dragged the scroll box/scroller about half way down the vertical scroll bar. The application determined that this distance and direction to scroll was -100 pixels.
- The application passed this distance to `ScrollRect`, which shifted the bits in the window 100 pixels upwards and created an update region in the vacated area of the window.
- The application passed the vertical scroll bar's current setting (100) in a parameter to `SetOrigin` so that the document's local coordinates were used when the update region of the window was redrawn. This changed the window's origin to (100,0).
- The application drew the text in the update region.
- The application reset the window's origin to (0,0).

Alternative to SetOrigin

There are alternatives to the `SetOrigin` methodology. `SetOrigin` simply helps you to offset the window's origin by the scroll bar's current settings when you update the window so that you can locate objects in a document using a coordinate system where the upper-left corner of the document is always at (0,0).

As an alternative to this approach, your application can leave the upper-left corner of the window at (0,0) and instead offset the items in your document, using `OffsetRect`, by an amount equal to the scroll bar's settings.

Scrolling a TextEdit Document and Scrolling Using the List Manager

TextEdit is a collection of functions and data structures which you can use to provide your application with basic text editing capabilities. Chapter 21 addresses the scrolling of TextEdit documents.

For scrolling lists of graphic or textual information, your application can use the List Manager to implement scroll bars (see Chapter 22).

Small Versions of Controls

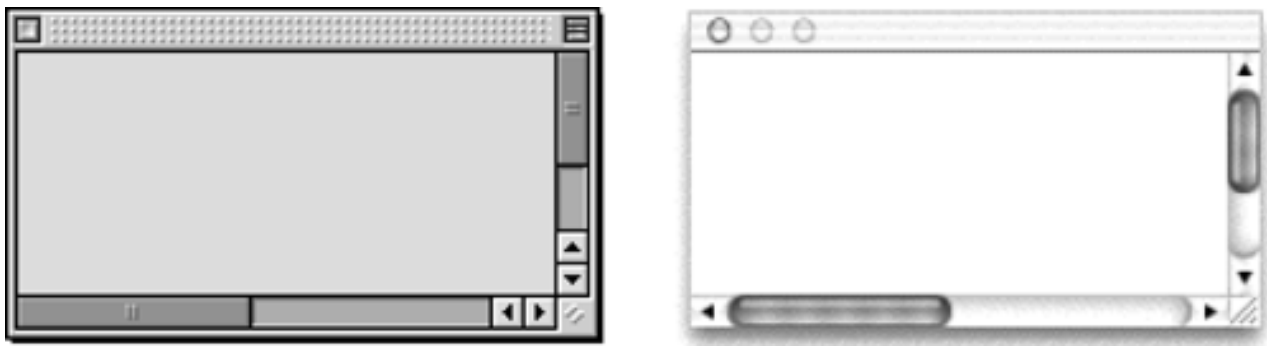
Human Interface Guidelines permit the use of small versions of certain controls, which should only be used when space is at a premium. Full size and small controls should not be mixed in the same window.

The following lists those controls described in this chapter that are available in small versions, and describes how to create them.

Control	*Mac OS X*	*Mac OS 8/9*
Push button	Make the control's rectangle 17 pixels high and call `SetControlFontStyle` to set small system font.	Make the control's rectangle 17 pixels high and use `SetControlFontStyle` to set the small system font.
Radio button Checkbox	Call `SetControlData` with the `kControlSizeTag` tag to make the control proper small, and call `SetControlFontStyle` to set the title to the small system font.	(Not available.)
Pop-up menu button	Create the control from a `'CNTL'` resource, specifying the `kControlPopupUseWFontVariant`, and set the owner window's font the small system font.	Create the control from a `'CNTL'` resource, specifying the `kControlPopupUseWFontVariant`, and set the owner window's font to the small system font.
Scroll bar	Call `SetControlData` with the `kControlSizeTag` tag to make the control small.	Make the control's rectangle 11 pixels wide (vertical scroll make bars) or high (horizontal scroll bars.

Small scroll bars should only be used in floating windows. Fig 12 shows an example.

Figure 12. SMALL SCROLL BARS IN A UTILITY (FLOATING) WINDOW



Main Control Manager Constants, Data Types and Functions Relevant to the Basic Controls, Primary Group Box and User Panes

Constants

Control Definition IDs

```
kControlPushButtonProc              = 368
kControlPushButLeftIconProc         = 374
kControlPushButRightIconProc        = 375
kControlCheckBoxProc                = 369
kControlCheckBoxAutoToggleProc      = 371
kControlRadioButtonProc             = 370
kControlRadioButtonAutoToggleProc   = 372
kControlScrollBarProc               = 384
kControlScrollBarLiveProc           = 386
kControlPopupButtonProc             = 400
kControlGroupBoxTextTitleProc       = 160
kControlUserPaneProc                = 256
```

Push Button Icon Alignment

```
kControlPushButtonIconOnLeft        = 6
kControlPushButtonIconOnRight       = 7
```

Pop-up Menu Button Variants

```
kControlPopupFixedWidthVariant      = 1 << 0
kControlPopupVariableWidthVariant   = 1 << 1
kControlPopupUseAddResMenuVariant   = 1 << 2
kControlPopupUseWFontVariant        = 1 << 3
```

Pop-up Title Characteristics

```
popupTitleBold                      = 1 << 8
popupTitleItalic                    = 1 << 9
popupTitleUnderline                 = 1 << 10
popupTitleOutline                   = 1 << 11
popupTitleShadow                    = 1 << 12
popupTitleCondense                  = 1 << 13
popupTitleExtend                    = 1 << 14
```

```
popupTitleNoStyle                    = 1 << 15
```

Control Variants

```
kControlNoVariant                    = 0
kControlUsesOwningWindowsFontVariant = 1 << 3
```

Control Part Codes

```
kControlNoPart                       = 0
kControlEntireControl                = 0
kControlLabelPart                    = 1
kControlMenuPart                     = 2
kControlCheckBoxPart                 = 11
kControlRadioButtonPart              = 11
kControlUpButtonPart                 = 20
kControlDownButtonPart               = 21
kControlPageUpPart                   = 22
kControlPageDownPart                 = 23
kControlIndicatorPart                = 129
kControlDisabledPart                 = 254
kControlInactivePart                 = 255
```

Checkbox Value Constants

```
kControlCheckBoxUncheckedValue       = 0
kControlCheckBoxCheckedValue         = 1
kControlCheckBoxMixedValue           = 2
```

Radio Button Value Constants

```
kControlRadioButtonUncheckedValue    = 0
kControlRadioButtonCheckedValue      = 1
kControlRadioButtonMixedValue        = 2
```

Control Data Tag Constants

```
kControlPushButtonDefaultTag         = FOUR_CHAR_CODE('dflt')
kControlPushButtonCancelTag          = FOUR_CHAR_CODE('cncl')
kControlPopupButtonMenuRefTag        = FOUR_CHAR_CODE('mhan')
kControlPopupButtonMenuIDTag         = FOUR_CHAR_CODE('mnid')
kControlPopupButtonExtraHeightTag    = FOUR_CHAR_CODE('exht')
kControlGroupBoxTitleRectTag         = FOUR_CHAR_CODE('trec')
```

Control Data Tag Constants (Mac OS X Only)

```
kControlKindTag                         = FOUR_CHAR_CODE('kind')
kControlSizeTag                         = FOUR_CHAR_CODE('size')
```

Control Font Style Flag Constants

```
kControlUseFontMask                     = 0x0001
kControlUseFaceMask                     = 0x0002
kControlUseSizeMask                     = 0x0004
kControlUseForeColorMask                = 0x0008
kControlUseBackColorMask                = 0x0010
kControlUseModeMask                     = 0x0020
kControlUseJustMask                     = 0x0040
kControlUseAllMask                      = 0x00FF
kControlAddFontSizeMask                 = 0x0100
```

Meta Font Constants

```
kControlFontBigSystemFont               = -1
kControlFontSmallSystemFont             = -2
kControlFontSmallBoldSystemFont         = -3
kControlFontViewSystemFont              = -4
```

Push Button Icon Alignment

```
kControlPushButtonIconOnLeft            = 6
kControlPushButtonIconOnRight           = 7
```

Control Image Content

```
kControlContentIconSuiteRes             = 1
kControlContentCIconRes                 = 2
kControlContentICONRes                  = 4
kControlContentIconSuiteHandle          = 129
kControlContentCIconHandle              = 130
kControlContentPictHandle               = 131
kControlContentIconRef                  = 132
kControlContentICON                     = 133
```

Control Kind (Mac OS X Only)

```
kControlKindPushButton                  = 'push'
kControlKindPushIconButton              = 'picn'
```

```
kControlKindCheckBox                = 'cbox'
kControlKindRadioButton             = 'rdio'
kControlKindScrollBar               = 'sbar'
kControlKindPopupButton             = 'popb'
kControlKindGroupBox                = 'grpb'
kControlKindUserPane                = 'upan'
```

Data Types

```
typedef struct OpaqueControlRef*  ControlRef;
typedef ControlRef  ControlHandle;
typedef SInt16      ControlPartCode;
typedef UInt16      ControlPushButtonIconAlignment;
typedef SInt16      ControlContentType;
```

Control Button Content Info

```
struct ControlButtonContentInfo
{
  ControlContentType  contentType;
  union
  {
    SInt16      resID;
    CIconHandle cIconHandle;
    Handle      iconSuite;
    IconRef     iconRef;
    PicHandle   picture;
    Handle      ICONHandle;
  }  u;
} ;
typedef struct ControlButtonContentInfo ControlButtonContentInfo;
typedef ControlButtonContentInfo *ControlButtonContentInfoPtr;
```

Control Font Style

```
struct ControlFontStyleRec
{
  SInt16   flags;
  SInt16   font;
  SInt16   size;
  SInt16   style;
  SInt16   mode;
  SInt16   just;
  RGBColor foreColor;
  RGBColor backColor;
} ;
typedef struct ControlFontStyleRec ControlFontStyleRec;
typedef ControlFontStyleRec *ControlFontStylePtr;
```

Control ID

```
struct ControlID
{
  OSType signature;
  SInt32 id;
} ;
typedef struct ControlIDControlID;
```

ControlKind

```
struct ControlKind
{
  OSType signature;
  OSType kind;
} ;
typedef struct ControlKind ControlKind;
```

Functions

Creating Controls

```
ControlRef  GetNewControl(SInt16 controlID,WindowPtr owner);
ControlRef  NewControl(WindowPtr owningWindow,const Rect *boundsRect,ConstStr255Param
            title,Boolean initiallyVisible,SInt16 initialValue,SInt16 minimumValue,
            SInt16 maximumValue,SInt16 procID,SInt32 controlReference);
OSStatus    CreatePushButtonControl(WindowRef window,const Rect *boundsRect,
            ConstStr255Param title,ControlRef *outControl);
OSStatus    CreatePushButtonWithIconControl(WindowRef window,const Rect *boundsRect,
            ConstStr255Param title,ControlButtonContentInfo *icon,
            ControlPushButtonIconAlignment iconAlignment,ControlRef *outControl);
OSStatus    CreateRadioButtonControl(WindowRef window, const Rect *boundsRect,
            ConstStr255Param title,Boolean autoToggle,ControlRef *outControl);
OSStatus    CreateCheckBoxControl(WindowRef window,const Rect *boundsRect,
            ConstStr255Param title,Boolean autoToggle,ControlRef *outControl);
OSStatus    CreateScrollBarControl(WindowRef window,Rect *boundsRect,
            SInt32 value,SInt32 minimum,SInt32 maximum,SInt32 viewSize,
            Boolean liveTracking,ControlActionUPP liveTrackingProc,
            ControlRef *outControl);
OSStatus    CreatePopupButtonControl(WindowRef window,const Rect *boundsRect,
            ConstStr255Param title,SInt16 menuID,Boolean variableWidth,SInt16 titleWidth,
            SInt16 titleJustification,Style titleStyle,ControlRef *outControl);
OSStatus    CreateGroupBoxControl(WindowRef window,const Rect *boundsRect,
            ConstStr255Param title,Boolean primary,ControlRef *outControl);
OSStatus    CreateUserPaneControl(WindowRef window,const Rect *boundsRect,
            UInt32 features,ControlRef *outControl);
```

Removing Controls

```
void        DisposeControl(ControlRef theControl);
void        KillControls(WindowPtr theWindow);
```

Embedding Controls

```
OSErr       CreateRootControl(WindowPtr inWindow,ControlRef *outControl);
OSErr       GetRootControl(WindowPtr inWindow,ControlRef *outControl);
OSErr       EmbedControl(ControlRef inControl,ControlRef inContainer);
OSErr       AutoEmbedControl(ControlRef inControl,WindowPtr inWindow);
OSErr       CountSubControl(ControlRef inControl,SInt16 *outNumChildren);
OSErr       GetIndexedSubControl(ControlRef inControl,SInt16 inIndex,
            ControlRef * outSubControl);
OSErr       GetSuperControl(ControlRef inControl,ControlRef *outParent);
OSErr       SetControlSupervisor(ControlRef inControl,ControlRef inBoss);
OSErr       DumpControlHierarchy(WindowPtr inWindow,const FSSpec *inDumpFile);
```

Displaying and Manipulating Controls

```
void        MoveControl(ControlRef theControl,SInt16 h,SInt16 v);
void        SizeControl(ControlRef theControl,SInt16 w,SInt16 h);
void        UpdateControls(WindowPtr theWindow,RgnHandle updateRgn);
void        DrawControls(WindowPtr theWindow);
void        DrawOneControl(ControlRef theControl);
void        DrawControlInCurrentPort(ControlRef inControl);
Boolean     IsControlActive (ControlRef inControl);
OSErr       ActivateControl (ControlRef inControl);
OSErr       DeactivateControl(ControlRef inControl);
Boolean     IsControlVisible(ControlRef inControl);
void        HideControl(ControlRef theControl);
```

```
void        ShowControl(ControlRef theControl);
OSErr       SetControlVisibility (ControlRef inControl,Boolean inIsVisible,
            Boolean inDoDraw);
OSErr       SendControlMessage(ControlRef inControl,SInt16 inMessage,SInt32 inParam);
void        DragControl(ControlRef theControl,Point startPt,const Rect *limitRect,
            const Rect *slopRect,DragConstraint axis);
Boolean     IsControlHilited(ControlRef control);
void        HiliteControl(ControlRef theControl,ControlPartCode hiliteState);
```

Enabling and Disabling Controls (Mac OS X Only)

```
OSStatus    EnableControl(ControlRef inControl);
OSStatus    DisableControl(ControlRef inControl);
Boolean     IsControlEnabled(ControlRef inControl);
```

Handling Events in Controls

```
ControlPartCode  FindControl(Point thePoint,WindowPtr theWindow,ControlRef *theControl);
ControlRef       FindControlUnderMouse(Point inWhere,WindowPtr inWindow,
                 SInt16 *outPart);
ControlPartCode  TrackControl(ControlRef theControl,Point thePoint,
                 ControlActionUPP actionProc);
SInt16           HandleControlClick(ControlRef inControl,Point inWhere,SInt16 inModifiers,
                 ControlActionUPP nAction);
ControlPartCode  TestControl(ControlRef theControl, PointthePt);
```

Accessing and Changing Control Settings and Data

```
WindowPtr   GetControlOwner(ControlRef control);
void        SetControlOwner(ControlRef control,WindowPtr owner);
Rect *      GetControlBounds(ControlRef control,Rect *bounds);
void        SetControlBounds(ControlRef control,const Rect *bounds);
Boolean     IsControlHilited(ControlRef control);
UInt16      GetControlHilite(ControlRef control);
Handle      GetControlDefinition(ControlRef control);
Handle      GetControlDataHandle(ControlRef control);
void        SetControlDataHandle(ControlRef control,Handle dataHandle);
MenuHandle  GetControlPopupMenuHandle(ControlRef control);
void        SetControlPopupMenuHandle(ControlRef control,MenuHandle popupMenu);
short       GetControlPopupMenuID(ControlRef control);
void        SetControlPopupMenuID(ControlRef control,short menuID);
SInt16      GetControlValue(ControlRef theControl);
void        SetControlValue(ControlRef theControl,SInt16 theValue);
SInt16      GetControlMinimum(ControlRef theControl);
void        SetControlMinimum(ControlRef theControl,SInt16 newMinimum);
SInt16      GetControlMaximum(ControlRef theControl);
void        SetControlMaximum(ControlRef theControl,SInt16 newMaximum);
SInt32      GetControl32BitValue(ControlRef theControl);
void        SetControl32BitValue(ControlRef theControl,SInt32 newValue);
SInt32      GetControl32BitMaximum(ControlRef theControl);
void        SetControl32BitMaximum(ControlRef theControl,SInt32 newMaximum);
SInt32      GetControl32BitMinimum(ControlRef theControl);
void        SetControl32BitMinimum(ControlRef theControl,SInt32 newMinimum);
void        GetControlTitle(ControlRef theControl,Str255 title);
void        SetControlTitle(ControlRef theControl,ConstStr255Param title);
OSStatus    SetControlTitleWithCFString(ControlRef inControl,CFStringRef inString);
SInt32      GetControlReference(ControlRef theControl);
void        SetControlReference(ControlRef theControl,SInt32 data);
ControlActionUPP  GetControlAction(ControlRef theControl);
void        SetControlAction(ControlRef theControl,ControlActionUPP actionProc);
SInt32      GetControlViewSize(ControlRef theControl);
void        SetControlViewSize(ControlRef theControl,SInt32 newViewSize);
SInt16      GetControlVariant(ControlRef theControl);
OSErr       SetControlData(ControlRef inControl,ControlPartCode inPart,
```

```
            ResType inTagName,Size inSize,Ptr inData);
OSErr       GetControlData (ControlRef inControl,ControlPartCode inPart,
            ResType inTagName,Size inBufferSize,Ptr inBuffer,Size *outActualSize);
OSErr       GetControlDataSize(ControlRef inControl,ControlPartCode inPart,
            ResType inTagName,Size *outMaxSize);
OSErr       SetControlVisibility(ControlRef inControl,Boolean inIsVisible,
            Boolean inDoDraw);
```

Setting the Control Font Style

```
OSErr       SetControlFontStyle(ControlRef inControl,const ControlFontStyleRec *inStyle);
```

Setting and Getting Control IDs

```
OSStatus    SetControlID(ControlRef inControl,const ControlID *inID);
OSStatus    GetControlID(ControlRef inControl,ControlID *outID);
OSStatus    GetControlByID(WindowRef inWindow,const ControlID *inID,ControlRef *outControl);
```

Getting Control Kind (Mac OS X Only)

```
OSStatus    GetControlKind(ControlRef inControl,ControlKind *outControlKind);
```

Creating and Disposing of Universal Procedure Pointers for Control Action Functions

```
ControlActionUPP  NewControlActionUPP(ControlActionProcPtr userRoutine);
void        DisposeControlActionUPP(ControlActionUPP userUPP);
```

Application-Defined (Callback) Function

```
void        myControlActionFunction(ControlRef theControl,ControlPartCode partCode);
```

Demonstration Program Controls1 Listing

```
// ****************************************************************************************
// Controls1.c                                                          CLASSIC EVENT MODEL
// ****************************************************************************************
//
// This program opens a kWindowFullZoomGrowDocumentProc window containing:
//
// •  Three pop-up menu buttons (fixed width, variable width and use window font variants).
//
// •  Three non-auto-toggling radio buttons auto-embedded in a primary group box (text title
//    variant).
//
// •  Three non-auto-toggling checkboxes auto-embedded in a primary group box (text title
//    variant).
//
// •  Four push buttons (two basic, one left color icon variant, and one right color icon
//     variant).
//
// •  A vertical scroll bar (non live-feedback variant) and a horizontal scroll bar (live-
//    feedback variant).
//
// Some controls are created using 'CNTL' resources. Others are created programmatically.
//
// The window also contains a window header frame in which is displayed:
//
// •  The menu items chosen from the pop-up menus.
//
// •  The identity of a push button when that push button is clicked.
//
// •  Scroll bar control values when the scroll arrows or gray areas/tracks of the scroll bars
//    are clicked and when the scroll box/scroller is dragged.
//
// The scroll bars are moved and resized when the user resizes or zooms the window; however,
// the scroll bars do not scroll the window contents.
//
// A Demonstration menu allows the user to deactivate the group boxes in which the radio
// buttons and checkboxes are embedded.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  An 'MBAR' resource, and 'MENU' resources for Apple, File, Edit, and Demonstration menus,
//    and the pop-up menus (preload, non-purgeable).
//
// •  A 'WIND' resource (purgeable) (initially not visible).
//
// •  'CNTL' resources for those controls not created programmatically.
//
// •  Two 'cicn' resources (purgeable) for the color icon variant buttons.
//
// •  An 'hrct' resource and an 'hwin' resource (both purgeable), which provide help balloons
//    describing  the various controls.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents,  canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// ****************************************************************************************

// ..................................................................... includes

#include <Carbon.h>
#include <string.h>
```

```
// ..................................................................................... defines

#define rMenubar          128
#define rWindow           128
#define mAppleApplication 128
#define  iAbout           1
#define mFile             129
#define  iQuit            12
#define mDemonstration    131
#define  iColor          1
#define  iGrids           2
#define cPopupFixed       128
#define cPopupWinFont     129
#define cRadiobuttonRed   130
#define cRadiobuttonBlue  131
#define cCheckboxGrid     132
#define cCheckboxGridsnap 133
#define cGroupBoxColor   134
#define cButton           135
#define cButtonLeftIcon   136
#define cScrollbarVert    137
#define kScrollBarWidth   15
#define MAX_UINT32        0xFFFFFFFF
#define MIN(a,b)          ((a) < (b) ? (a) : (b))

// .....................................................................................
typedefs

typedef struct
{
  ControlRef popupFixedRef;
  ControlRef popupVariableRef;
  ControlRef popupWinFontRef;
  ControlRef groupboxColorRef;
  ControlRef groupboxGridsRef;
  ControlRef buttonRef;
  ControlRef buttonDefaultRef;
  ControlRef buttonLeftIconRef;
  ControlRef buttonRightIconRef;
  ControlRef radiobuttonRedRef;
  ControlRef radiobuttonWhiteRef;
  ControlRef radiobuttonBlueRef;
  ControlRef checkboxGridRef;
  ControlRef checkboxRulersRef;
  ControlRef checkboxGridSnapRef;
  ControlRef scrollbarVertRef;
  ControlRef scrollbarHorizRef;
}  docStruc;

typedef docStruc **docStrucHandle;

// ..................................................................... global
variables

ControlActionUPP gActionFunctionVertUPP;
ControlActionUPP gActionFunctionHorizUPP;
Boolean          gRunningOnX = false;
WindowRef        gWindowRef;
Boolean          gDone;
Boolean          gInBackground = false;
Str255           gCurrentString;

// ..................................................................... function
prototypes

void  main               (void);
void  doPreliminaries    (void);
```

```
OSErr quitAppEventHandler (AppleEvent *,AppleEvent *,SInt32);
void  doGetControls       (WindowRef);
void  doEvents            (EventRecord *);
void  doMouseDown         (EventRecord *);
void  doMenuChoice        (SInt32);
void  doUpdate            (EventRecord *);
void  doActivate          (EventRecord *);
void  doActivateWindow    (WindowRef,Boolean);
void  doOSEvent           (EventRecord *);
void  doInContent         (EventRecord *,WindowRef);
void  doPopupMenuChoice   (WindowRef,ControlRef,SInt16);
void  doVertScrollbar     (ControlPartCode,WindowRef,ControlRef,Point);
void  actionFunctionVert  (ControlRef,ControlPartCode);
void  actionFunctionHoriz (ControlRef,ControlPartCode);
void  doMoveScrollBox     (ControlRef,SInt16);
void  doRadioButtons      (ControlRef,WindowRef);
void  doCheckboxes        (ControlRef);
void  doPushButtons       (ControlRef,WindowRef);
void  doAdjustScrollBars  (WindowRef);
void  doDrawMessage       (WindowRef,Boolean);
void  doConcatPStrings    (Str255,Str255);
void  doCopyPString       (Str255,Str255);
void  helpTags            (WindowRef);

// ********************************************************************************** main

void  main(void)
{
  MenuBarHandle  menubarHdl;
  SInt32         response;
  MenuRef        menuRef;
  docStrucHandle docStrucHdl;
  EventRecord    EventStructure;

  // ..................................................................... do preliminaries

  doPreliminaries();

  // ................................................... create universal procedure pointers

  gActionFunctionVertUPP  = NewControlActionUPP((ControlActionProcPtr) actionFunctionVert);
  gActionFunctionHorizUPP = NewControlActionUPP((ControlActionProcPtr) actionFunctionHoriz);

  // ..................................................... set up menu bar and menus

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    ExitToShell();
  SetMenuBar(menubarHdl);
  DrawMenuBar();

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
      DisableMenuItem(menuRef,0);
    }

    gRunningOnX = true;
  }

  // ..................................... initial advisory text for window header
```

```
  doCopyPString("\pBalloon and Help tag help is available",gCurrentString);

  // ........ open a window, set font size, set Appearance-compliant color/pattern for window

  if(!(gWindowRef = GetNewCWindow(rWindow,NULL,(WindowRef) -1)))
    ExitToShell();

  SetPortWindowPort(gWindowRef);
  UseThemeFont(kThemeSmallSystemFont,smSystemScript);

  SetThemeWindowBackground(gWindowRef,kThemeBrushDialogBackgroundActive,false);

  // ........... get block for document structure, assign handle to window record refCon field

  if(!(docStrucHdl = (docStrucHandle) NewHandle(sizeof(docStruc))))
    ExitToShell();

  SetWRefCon(gWindowRef,(SInt32) docStrucHdl);

  // ....................... get controls, adjust scroll bars, get help tags, and show window

  doGetControls(gWindowRef);
  doAdjustScrollBars(gWindowRef);

  if(gRunningOnX)
    helpTags(gWindowRef);

  ShowWindow(gWindowRef);

// ..................................................................... enter eventLoop

  gDone = false;

  while(!gDone)
  {
    if(WaitNextEvent(everyEvent,&EventStructure,MAX_UINT32,NULL))
      doEvents(&EventStructure);
  }
}

// ********************************************************************** doPreliminaries

void  doPreliminaries(void)
{
  OSErr osError;
  MoreMasterPointers(128);
  InitCursor();
  FlushEvents(everyEvent,0);

  osError = AEInstallEventHandler(kCoreEventClass,kAEQuitApplication,
                           NewAEEventHandlerUPP((AEEventHandlerProcPtr) quitAppEventHandler),
                           0L,false);
  if(osError != noErr)
    ExitToShell();
}

// ********************************************************************** doQuitAppEvent

OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  OSErr    osError;
  DescType returnedType;
  Size     actualSize;

  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,NULL,0,
                              &actualSize);

  if(osError == errAEDescNotFound)
```

```
  {
    gDone = true;
    osError = noErr;
  }
  else if(osError == noErr)
    osError = errAEParamMissed;

  return osError;
}

// ********************************************************************************* doGetControls

void  doGetControls(WindowRef windowRef)
{
  ControlRef               controlRef;
  docStrucHandle           docStrucHdl;
  OSStatus                 osError;
  Rect                     popupVariableRect    = {  73,  25, 93,245 } ;
  Rect                     radioButtonWhiteRect = {  183, 35,201, 92 } ;
  Rect                     checkboxRulersRect   = {  183,138,201,242 } ;
  Rect                     groupboxGridsRect    = {  136,123,236,252 } ;
  Rect                     buttonDefaultRect    = {  252,141,272,210 } ;
  Rect                     buttonRightIconRect  = {  285,141,305,220 } ;
  Rect                     scrollbarVertRect    = {  0,    0, 16,100 } ;
  ControlButtonContentInfo buttonContentInfo;
  Boolean                  booleanData = true;
  ControlFontStyleRec      controlFontStyleStruc;

  if(!gRunningOnX)
    CreateRootControl(windowRef,&controlRef);

  docStrucHdl = (docStrucHandle) (GetWRefCon(windowRef));
  // ..................................................................... popup menu buttons

  if(!((*docStrucHdl)->popupFixedRef = GetNewControl(cPopupFixed,windowRef)))
    ExitToShell();

  if((osError = CreatePopupButtonControl(windowRef,&popupVariableRect,CFSTR("Time Zone:"),
                                         132,true,76,popupTitleLeftJust,popupTitleNoStyle,
                                         &(*docStrucHdl)->popupVariableRef)) == noErr)
    ShowControl((*docStrucHdl)->popupVariableRef);
  else
    ExitToShell();

  if(!((*docStrucHdl)->popupWinFontRef = GetNewControl(cPopupWinFont,windowRef)))
    ExitToShell();

  // ..................................................................... radio buttons

  if(!((*docStrucHdl)->radiobuttonRedRef = GetNewControl(cRadiobuttonRed,windowRef)))
    ExitToShell();

  if((osError = CreateRadioButtonControl(windowRef,&radioButtonWhiteRect,CFSTR("White"),0,
                                        false,&(*docStrucHdl)->radiobuttonWhiteRef)) == noErr)
    ShowControl((*docStrucHdl)->radiobuttonWhiteRef);
  else
    ExitToShell();

  if(!((*docStrucHdl)->radiobuttonBlueRef = GetNewControl(cRadiobuttonBlue,windowRef)))
    ExitToShell();

  // ..................................................................... checkboxes

  if(!((*docStrucHdl)->checkboxGridRef = GetNewControl(cCheckboxGrid,windowRef)))
    ExitToShell();

  if((osError = CreateCheckBoxControl(windowRef,&checkboxRulersRect,CFSTR("Rulers"),0,false,
                                      &(*docStrucHdl)->checkboxRulersRef)) == noErr)
```

```
    ShowControl((*docStrucHdl)->checkboxRulersRef);
  else
    ExitToShell();

  if(!((*docStrucHdl)->checkboxGridSnapRef = GetNewControl(cCheckboxGridsnap,windowRef)))
    ExitToShell();

  // ..................................................................... group boxes

  if(!((*docStrucHdl)->groupboxColorRef = GetNewControl(cGroupBoxColor,windowRef)))
    ExitToShell();

  if((osError = CreateGroupBoxControl(windowRef,&groupboxGridsRect,CFSTR("Grids"),true,
                                      &(*docStrucHdl)->groupboxGridsRef)) == noErr)
    ShowControl((*docStrucHdl)->groupboxGridsRef);
  else
    ExitToShell();
  // ..................................................................... push
buttons

  if(!((*docStrucHdl)->buttonRef = GetNewControl(cButton,windowRef)))
    ExitToShell();

  if((osError = CreatePushButtonControl(windowRef,&buttonDefaultRect,CFSTR("OK"),
                                        &(*docStrucHdl)->buttonDefaultRef)) == noErr)
    ShowControl((*docStrucHdl)->buttonDefaultRef);
  else
    ExitToShell();

  if(!((*docStrucHdl)->buttonLeftIconRef = GetNewControl(cButtonLeftIcon,windowRef)))
    ExitToShell();

  buttonContentInfo.contentType = kControlContentCIconRes;
  buttonContentInfo.u.resID = 101;
  if((osError = CreatePushButtonWithIconControl(windowRef,&buttonRightIconRect,
                                                CFSTR("Button"),&buttonContentInfo,
                                                kControlPushButtonIconOnRight,
                                                &(*docStrucHdl)->buttonRightIconRef)) == noErr)
    ShowControl((*docStrucHdl)->buttonRightIconRef);
  else
    ExitToShell();

  // ..................................................................... scroll bars

  if(!((*docStrucHdl)->scrollbarVertRef = GetNewControl(cScrollbarVert,windowRef)))
    ExitToShell();

  if((osError = CreateScrollBarControl(windowRef,&scrollbarVertRect,0,0,100,100,true,
                                       gActionFunctionHorizUPP,
                                       &(*docStrucHdl)->scrollbarHorizRef)) == noErr)
    ShowControl((*docStrucHdl)->scrollbarHorizRef);
  else
    ExitToShell();

  // ..........................................................................

  AutoEmbedControl((*docStrucHdl)->radiobuttonRedRef,windowRef);
  AutoEmbedControl((*docStrucHdl)->radiobuttonWhiteRef,windowRef);
  AutoEmbedControl((*docStrucHdl)->radiobuttonBlueRef,windowRef);
  AutoEmbedControl((*docStrucHdl)->checkboxGridRef,windowRef);
  AutoEmbedControl((*docStrucHdl)->checkboxRulersRef,windowRef);
  AutoEmbedControl((*docStrucHdl)->checkboxGridSnapRef,windowRef);

  SetControlData((*docStrucHdl)->buttonDefaultRef,kControlEntireControl,
                 kControlPushButtonDefaultTag,sizeof(booleanData),&booleanData);

  controlFontStyleStruc.flags = kControlUseFontMask;
  controlFontStyleStruc.font = kControlFontSmallSystemFont;
```

```
  SetControlFontStyle((*docStrucHdl)->buttonLeftIconRef,&controlFontStyleStruc);
  controlFontStyleStruc.font = kControlFontSmallBoldSystemFont;
  SetControlFontStyle((*docStrucHdl)->buttonRightIconRef,&controlFontStyleStruc);

  DeactivateControl((*docStrucHdl)->checkboxRulersRef);
}
// ********************************************************************************** doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
  SInt32        menuChoice;
  MenuID        menuID;
  MenuItemIndex menuItem;

  switch(eventStrucPtr->what)
  {
    case kHighLevelEvent:
      AEProcessAppleEvent(eventStrucPtr);
      break;

    case keyDown:
      if((eventStrucPtr->modifiers & cmdKey) != 0)
      {
        menuChoice = MenuEvent(eventStrucPtr);
        menuID = HiWord(menuChoice);
        menuItem = LoWord(menuChoice);
        if(menuID == mFile && menuItem  == iQuit)
          gDone = true;
      }
      break;

    case mouseDown:
      doMouseDown(eventStrucPtr);
      break;

    case updateEvt:
      doUpdate(eventStrucPtr);
      break;

    case activateEvt:
      doActivate(eventStrucPtr);
      break;

    case osEvt:
      doOSEvent(eventStrucPtr);
      break;
  }
}

// ********************************************************************************** doMouseDown

void  doMouseDown(EventRecord *eventStrucPtr)
{
  WindowPartCode partCode, zoomPart;
  WindowRef      windowRef;
  Rect           constraintRect, mainScreenRect, portRect;
  BitMap         screenBits;
  Point          standardStateHeightAndWidth;
  partCode = FindWindow(eventStrucPtr->where,&windowRef);

  switch(partCode)
  {
    case inMenuBar:
      doMenuChoice(MenuSelect(eventStrucPtr->where));
      break;

    case inContent:
      if(windowRef != FrontWindow())
```

```
        SelectWindow(windowRef);
      else
        doInContent(eventStrucPtr,windowRef);
      break;

    case inDrag:
      DragWindow(windowRef,eventStrucPtr->where,NULL);
      break;

    case inGoAway:
      if(TrackGoAway(windowRef,eventStrucPtr->where) == true)
        gDone = true;
      break;

    case inGrow:
      constraintRect.top   = 341;
      constraintRect.left = 287;
      constraintRect.bottom = constraintRect.right = 32767;
      ResizeWindow(windowRef,eventStrucPtr->where,&constraintRect,NULL);
      doAdjustScrollBars(windowRef);
      doDrawMessage(windowRef,true);
      break;

    case inZoomIn:
    case inZoomOut:
      mainScreenRect = GetQDGlobalsScreenBits(&screenBits)->bounds;
      standardStateHeightAndWidth.v = mainScreenRect.bottom - 75;
      standardStateHeightAndWidth.h = 600;

      if(IsWindowInStandardState(windowRef,&standardStateHeightAndWidth,NULL))
        zoomPart = inZoomIn;
      else
        zoomPart = inZoomOut;

      if(TrackBox(windowRef,eventStrucPtr->where,partCode))
      {
        GetWindowPortBounds(windowRef,&portRect);
        EraseRect(&portRect);
        ZoomWindowIdeal(windowRef,zoomPart,&standardStateHeightAndWidth);
        doAdjustScrollBars(windowRef);
      }
      break;
  }
}
// ********************************************************************************* doMenuChoice

void  doMenuChoice(SInt32 menuChoice)
{
  MenuID         menuID;
  MenuItemIndex  menuItem;
  MenuRef        menuRef;
  WindowRef      windowRef;
  docStrucHandle docStrucHdl;

  menuID = HiWord(menuChoice);
  menuItem = LoWord(menuChoice);

  if(menuID == 0)
    return;

  switch(menuID)
  {
    case mAppleApplication:
      if(menuItem == iAbout)
        SysBeep(10);
      break;

    case mFile:
```

```
      if(menuItem == iQuit)
        gDone = true;
      break;

    case mDemonstration:
      menuRef = GetMenuRef(mDemonstration);
      windowRef = FrontWindow();
      docStrucHdl = (docStrucHandle) (GetWRefCon(windowRef));
      if(menuItem == iColor)
      {

        if(IsControlVisible((*docStrucHdl)->groupboxColorRef))
        {
          HideControl((*docStrucHdl)->groupboxColorRef);
          SetMenuItemText(menuRef,iColor,"\pShow Color");
        }
        else
        {
          ShowControl((*docStrucHdl)->groupboxColorRef);
          SetMenuItemText(menuRef,iColor,"\pHide Color");
        }
      }
      else if(menuItem == iGrids)
      {
        if(IsControlActive((*docStrucHdl)->groupboxGridsRef))
        {
          DeactivateControl((*docStrucHdl)->groupboxGridsRef);
          SetMenuItemText(menuRef,iGrids,"\pActivate Grids");
        }
        else
        {
          ActivateControl((*docStrucHdl)->groupboxGridsRef);
          SetMenuItemText(menuRef,iGrids,"\pDeactivate Grids");
        }
      }
      break;
  }

  HiliteMenu(0);
}

// ********************************************************************************** doUpdate

void  doUpdate(EventRecord *eventStrucPtr)
{
  WindowRef windowRef;
  RgnHandle regionHdl;

  windowRef = (WindowRef) eventStrucPtr->message;

  BeginUpdate(windowRef);

  SetPortWindowPort(windowRef);
  doDrawMessage(windowRef,!gInBackground);

  if(regionHdl = NewRgn())
  {
    GetPortVisibleRegion(GetWindowPort(windowRef),regionHdl);
    UpdateControls(windowRef,regionHdl);
    DisposeRgn(regionHdl);
  }

  EndUpdate(windowRef);
}

// ********************************************************************************** doActivate

void  doActivate(EventRecord *eventStrucPtr)
```

```
{
  WindowRef windowRef;
  Boolean   becomingActive;

  windowRef = (WindowRef) eventStrucPtr->message;
  becomingActive = ((eventStrucPtr->modifiers & activeFlag) == activeFlag);
  doActivateWindow(windowRef,becomingActive);
}

// ********************************************************************** doActivateWindow

void  doActivateWindow(WindowRef windowRef,Boolean becomingActive)
{
  ControlRef controlRef;
  GetRootControl(windowRef,&controlRef);

  if(becomingActive)
  {
    ActivateControl(controlRef);
    doDrawMessage(windowRef,becomingActive);
  }
  else
  {
    DeactivateControl(controlRef);
    doDrawMessage(windowRef,becomingActive);
  }
}

// ************************************************************************** doOSEvent

void  doOSEvent(EventRecord *eventStrucPtr)
{
  switch((eventStrucPtr->message >> 24) & 0x000000FF)
  {
    case suspendResumeMessage:
      if((eventStrucPtr->message & resumeFlag) == 1)
      {
        SetThemeCursor(kThemeArrowCursor);
        gInBackground = false;
      }
      else
        gInBackground = true;
      break;
  }
}

// ************************************************************************** doInContent

void  doInContent(EventRecord *eventStrucPtr,WindowRef windowRef)
{
  docStrucHandle docStrucHdl;
  ControlRef     controlRef;
  SInt16         controlValue, controlPartCode;

  docStrucHdl = (docStrucHandle) (GetWRefCon(windowRef));

  SetPortWindowPort(windowRef);
  GlobalToLocal(&eventStrucPtr->where);
  if(controlPartCode = FindControl(eventStrucPtr->where,windowRef,&controlRef))
  {

    if(controlRef == (*docStrucHdl)->popupFixedRef ||
       controlRef == (*docStrucHdl)->popupVariableRef ||
       controlRef == (*docStrucHdl)->popupWinFontRef)
    {
      TrackControl(controlRef,eventStrucPtr->where,(ControlActionUPP) -1);
      controlValue = GetControlValue(controlRef);
      doPopupMenuChoice(windowRef,controlRef,controlValue);
```

```
    }
    else if(controlRef == (*docStrucHdl)->scrollbarVertRef)
    {
      doVertScrollbar(controlPartCode,windowRef,controlRef,eventStrucPtr->where);
    }
    else if(controlRef == (*docStrucHdl)->scrollbarHorizRef)
    {
      TrackControl(controlRef,eventStrucPtr->where,gActionFunctionHorizUPP);
    }
    else
    {
      if(TrackControl(controlRef,eventStrucPtr->where,NULL))
      {
        if(controlRef == (*docStrucHdl)->radiobuttonRedRef ||
           controlRef == (*docStrucHdl)->radiobuttonWhiteRef ||
           controlRef == (*docStrucHdl)->radiobuttonBlueRef)
        {
          doRadioButtons(controlRef,windowRef);
        }
        if(controlRef == (*docStrucHdl)->checkboxGridRef ||
           controlRef == (*docStrucHdl)->checkboxRulersRef ||
           controlRef == (*docStrucHdl)->checkboxGridSnapRef)
        {
          doCheckboxes(controlRef);
        }
        if(controlRef == (*docStrucHdl)->buttonRef ||
           controlRef == (*docStrucHdl)->buttonDefaultRef ||
           controlRef == (*docStrucHdl)->buttonLeftIconRef ||
           controlRef == (*docStrucHdl)->buttonRightIconRef)
        {
          doPushButtons(controlRef,windowRef);
        }
      }
    }
  }
}

// ************************************************************************** doPopupMenuChoice

void  doPopupMenuChoice(WindowRef windowRef,ControlRef controlRef,SInt16 controlValue)
{
  MenuRef menuRef;
  Size    actualSize;

  GetControlData(controlRef,kControlEntireControl,kControlPopupButtonMenuHandleTag,
                 sizeof(menuRef),&menuRef,&actualSize);
  GetMenuItemText(menuRef,controlValue,gCurrentString);
  doDrawMessage(windowRef,true);
}

// ************************************************************************** doVertScrollbar

void  doVertScrollbar(ControlPartCode controlPartCode,WindowRef windowRef,
                      ControlRef controlRef,Point mouseXY)
{
  Str255 valueString;
  doCopyPString("\pScroll Bar Control Value: ",gCurrentString);

  switch(controlPartCode)
  {
    case kControlIndicatorPart:
      if(TrackControl(controlRef,mouseXY,NULL))
      {
        NumToString((SInt32) GetControlValue(controlRef),valueString);
        doConcatPStrings(gCurrentString,valueString);
        doDrawMessage(windowRef,true);
      }
      break;
```

```
    case kControlUpButtonPart:
    case kControlDownButtonPart:
    case kControlPageUpPart:
    case kControlPageDownPart:
      TrackControl(controlRef,mouseXY,gActionFunctionVertUPP);
      break;
  }
}

// ********************************************************************** actionFunctionVert

void  actionFunctionVert(ControlRef controlRef,ControlPartCode controlPartCode)
{
  SInt16    scrollDistance, controlValue;
  Str255    valueString;
  WindowRef windowRef;

  doCopyPString("\pScroll Bar Control Value: ",gCurrentString);

  if(controlPartCode)
  {
    switch(controlPartCode)
    {
      case kControlUpButtonPart:
      case kControlDownButtonPart:
        scrollDistance = 2;
        break;

      case kControlPageUpPart:
      case kControlPageDownPart:
        scrollDistance = 55;
        break;
    }

    if((controlPartCode == kControlDownButtonPart) ||
       (controlPartCode == kControlPageDownPart))
      scrollDistance = -scrollDistance;

    controlValue = GetControlValue(controlRef);
    if(((controlValue == GetControlMaximum(controlRef)) && scrollDistance < 0) ||
       ((controlValue == GetControlMinimum(controlRef)) && scrollDistance > 0))
      return;
    doMoveScrollBox(controlRef,scrollDistance);

    NumToString((SInt32) GetControlValue(controlRef),valueString);
    doConcatPStrings(gCurrentString,valueString);
    windowRef = GetControlOwner(controlRef);
    doDrawMessage(windowRef,true);
  }
}

// ********************************************************************** actionFunctionHoriz

void  actionFunctionHoriz(ControlRef controlRef,ControlPartCode controlPartCode)
{
  SInt16    scrollDistance, controlValue;
  Str255    valueString;
  WindowRef windowRef;

  doCopyPString("\pScroll Bar Control Value: ",gCurrentString);

  if(controlPartCode != kControlIndicatorPart)
  {
    switch(controlPartCode)
    {
      case kControlUpButtonPart:
      case kControlDownButtonPart:
```

```
        scrollDistance = 2;
        break;

      case kControlPageUpPart:
      case kControlPageDownPart:
        scrollDistance = 55;
        break;
    }

    if((controlPartCode == kControlDownButtonPart) ||
       (controlPartCode == kControlPageDownPart))
      scrollDistance = -scrollDistance;

    controlValue = GetControlValue(controlRef);
    if(((controlValue == GetControlMaximum(controlRef)) && scrollDistance < 0) ||
       ((controlValue == GetControlMinimum(controlRef)) && scrollDistance > 0))
      return;

    doMoveScrollBox(controlRef,scrollDistance);
  }

  NumToString((SInt32) GetControlValue(controlRef),valueString);
  doConcatPStrings(gCurrentString,valueString);
  windowRef = GetControlOwner(controlRef);
  doDrawMessage(windowRef,true);
}
// ********************************************************************** doMoveScrollBox

void doMoveScrollBox(ControlRef controlRef,SInt16 scrollDistance)
{
  SInt16 oldControlValue, controlValue, controlMax;

  oldControlValue = GetControlValue(controlRef);
  controlMax = GetControlMaximum(controlRef);

  controlValue = oldControlValue - scrollDistance;

  if(controlValue < 0)
    controlValue = 0;
  else if(controlValue > controlMax)
    controlValue = controlMax;

  SetControlValue(controlRef,controlValue);
}

// ********************************************************************** doRadioButtons

void  doRadioButtons(ControlRef controlRef,WindowRef windowRef)
{
  docStrucHandle docStrucHdl;

  docStrucHdl = (docStrucHandle) (GetWRefCon(windowRef));

  SetControlValue((*docStrucHdl)->radiobuttonRedRef,kControlRadioButtonUncheckedValue);
  SetControlValue((*docStrucHdl)->radiobuttonWhiteRef,kControlRadioButtonUncheckedValue);
  SetControlValue((*docStrucHdl)->radiobuttonBlueRef,kControlRadioButtonUncheckedValue);
  SetControlValue(controlRef,kControlRadioButtonCheckedValue);
}

// ********************************************************************** doCheckboxes

void  doCheckboxes(ControlRef controlRef)
{
  SetControlValue(controlRef,!GetControlValue(controlRef));
}

// ********************************************************************** doPushButtons
```

```
void  doPushButtons(ControlRef controlRef,WindowRef windowRef)
{
  docStrucHandle docStrucHdl;

  docStrucHdl = (docStrucHandle) (GetWRefCon(windowRef));

  if(controlRef == (*docStrucHdl)->buttonRef)
  {
    doCopyPString("\pButton",gCurrentString);
    doDrawMessage(windowRef,true);
  }
  else if(controlRef == (*docStrucHdl)->buttonDefaultRef)
  {
    doCopyPString("\pDefault Button",gCurrentString);
    doDrawMessage(windowRef,true);
  }
  else if(controlRef == (*docStrucHdl)->buttonLeftIconRef)
  {
    doCopyPString("\pLeft Icon Button",gCurrentString);
    doDrawMessage(windowRef,true);
  }
  else if(controlRef == (*docStrucHdl)->buttonRightIconRef)
  {
    doCopyPString("\pRight Icon Button",gCurrentString);
    doDrawMessage(windowRef,true);
  }
}

// ********************************************************************** doAdjustScrollBars

void  doAdjustScrollBars(WindowRef windowRef)
{
  Rect           portRect;
  docStrucHandle docStrucHdl;

  docStrucHdl = (docStrucHandle) (GetWRefCon(windowRef));

  GetWindowPortBounds(windowRef,&portRect);

  HideControl((*docStrucHdl)->scrollbarVertRef);
  HideControl((*docStrucHdl)->scrollbarHorizRef);

  MoveControl((*docStrucHdl)->scrollbarVertRef,portRect.right - kScrollBarWidth,
              portRect.top + 25);
  MoveControl((*docStrucHdl)->scrollbarHorizRef,portRect.left -1,
              portRect.bottom - kScrollBarWidth);

  SizeControl((*docStrucHdl)->scrollbarVertRef,16, portRect.bottom - 39);
  SizeControl((*docStrucHdl)->scrollbarHorizRef, portRect.right - 13,16);

  ShowControl((*docStrucHdl)->scrollbarVertRef);
  ShowControl((*docStrucHdl)->scrollbarHorizRef);

  SetControlMaximum((*docStrucHdl)->scrollbarVertRef,portRect.bottom - portRect.top - 25
                    - kScrollBarWidth);
  SetControlMaximum((*docStrucHdl)->scrollbarHorizRef,portRect.right - portRect.left
                    - kScrollBarWidth);
}

// ********************************************************************** doDrawMessage

void  doDrawMessage(WindowRef windowRef,Boolean inState)
{
  Rect        portRect, headerRect;
  CFStringRef stringRef;
  Rect        textBoxRect;

  if(windowRef == gWindowRef)
```

```
  {
    SetPortWindowPort(windowRef);

    GetWindowPortBounds(windowRef,&portRect);

    SetRect(&headerRect,portRect.left - 1,portRect.top - 1,portRect.right + 1,
          portRect.top + 26);
    DrawThemeWindowHeader(&headerRect,inState);

    if(inState == kThemeStateActive)
      TextMode(srcOr);
    else
      TextMode(grayishTextOr);

    stringRef = CFStringCreateWithPascalString(NULL,gCurrentString,
                                               kCFStringEncodingMacRoman);
    SetRect(&textBoxRect,portRect.left,6,portRect.right,21);
    DrawThemeTextBox(stringRef,kThemeSmallSystemFont,0,true,&textBoxRect,teJustCenter,NULL);
    if(stringRef != NULL)
      CFRelease(stringRef);

    TextMode(srcOr);
  }
}

// ************************************************************************** doConcatPStrings

void  doConcatPStrings(Str255 targetString,Str255 appendString)
{
  SInt16 appendLength;

  appendLength = MIN(appendString[0],255 - targetString[0]);

  if(appendLength > 0)
  {
    BlockMoveData(appendString+1,targetString+targetString[0]+1,(SInt32) appendLength);
    targetString[0] += appendLength;
  }
}

// ************************************************************************** doCopyPString

void  doCopyPString(Str255 sourceString,Str255 destinationString)
{
  SInt16 stringLength;

  stringLength = sourceString[0];
  BlockMove(sourceString + 1,destinationString + 1,stringLength);
  destinationString[0] = stringLength;
}
// ****************************************************************************** helpTags

void  helpTags(WindowRef windowRef)
{
  docStrucHandle   docStrucHdl;
  HMHelpContentRec helpContent;

  memset(&helpContent,0,sizeof(helpContent));

  docStrucHdl = (docStrucHandle) (GetWRefCon(windowRef));

  HMSetTagDelay(500);
  HMSetHelpTagsDisplayed(true);
  helpContent.version = kMacHelpVersion;
  helpContent.tagSide = kHMOutsideTopCenterAligned;

  helpContent.content[kHMMinimumContentIndex].contentType = kHMStringResContent;
  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmResID = 128;
```

```
  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 1;
  HMSetControlHelpContent((*docStrucHdl)->popupFixedRef,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 2;
  HMSetControlHelpContent((*docStrucHdl)->popupVariableRef,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 3;
  HMSetControlHelpContent((*docStrucHdl)->popupWinFontRef,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 4;
  HMSetControlHelpContent((*docStrucHdl)->groupboxColorRef,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 5;
  HMSetControlHelpContent((*docStrucHdl)->groupboxGridsRef,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 6;
  HMSetControlHelpContent((*docStrucHdl)->buttonRef,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 7;
  HMSetControlHelpContent((*docStrucHdl)->buttonDefaultRef,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 8;
  HMSetControlHelpContent((*docStrucHdl)->buttonLeftIconRef,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 9;
  HMSetControlHelpContent((*docStrucHdl)->buttonRightIconRef,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 10;
  HMSetControlHelpContent((*docStrucHdl)->scrollbarVertRef,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 11;
  HMSetControlHelpContent((*docStrucHdl)->scrollbarHorizRef,&helpContent);
}

// ********************************************************************************************
```

Demonstration Program Controls1 Comments

When this program is run, the user should

- On Mac OS 8/9, choose Show Balloons from the Help menu and peruse the help balloons which are invoked when the mouse cursor is moved over the various controls.
- On Mac OS X, peruse the Help tags which are invoked when the mouse cursor is held over the various controls.
- Choose items from each of the pop-up menu buttons, noting that the chosen item is displayed in the window header.
- Click on the radio buttons, checkboxes, and push buttons, noting particularly that the radio button settings are mutually exclusive and that checkbox settings are not.
- Click in the scroll bar arrows and gray areas/tracks of the scroll bars, noting the control value changes displayed in the window header.
- Drag the scroll box/scroller of the vertical scroll bar (which uses the non-live-feedback CDEF variant), noting that only a ghosted outline is dragged and that the control value does not change until the mouse button is released.
- Drag the scroll box/scroller of the horizontal scroll bar (which uses the live-feedback CDEF variant), noting that the scroll box proper is dragged and that the control value is continually updated during the drag.
- Resize and zoom the window, noting that the scroll bars are moved and resized in response to those actions and the change in the maximum value of the scroll bars.
- Send the program to the background and bring it to the foreground, noting the changes to the appearance of the controls. (The program activates and deactivates the root control only; however, because all controls are embedded in the root control, all controls are activated and deactivated along with the root control.)
- Alternately hide and show the Color primary group box by choosing the associated item in the Demonstration menu. (The program hides and shows the primary group box only; however, because the radio buttons are embedded in the primary group box, those controls hidden and shown along with the primary group box.)

- Alternately activate and deactivate the Grids primary group box by choosing the associated item in the Demonstration menu. (The program activates and deactivates the primary group box only; however, because the checkboxes are embedded in the primary group box, those controls are activated and deactivated along with the primary group box.) Also note the latency of the Show Rulers checkbox. It is deactivated at program launch, and retains that status when the primary group box is deactivated and then re-activated.

In this program (and all others that use Help tags), the header file string.h is included and the library MSL C.PPC.Lib has been added to the project because of the use of memset in the function helpTags.

If you want to display the Help tags, rather than Balloon help, on Mac OS 8/9

- Comment out the line "if(gRunningOnX)" before the line "helpTags (gWindowRef);".
- In the function helpTags, change "kHMOutsideTopCenterAligned" to "kHMOutsideLeftCenterAligned". (At the time of writing, use of any but the first thirteen positioning constants in MacHelp.h defeated the display of Help tags on Mac OS 8/9.)

Help tag creation is addressed in Chapter 25.

defines

Constants are established for `'CNTL'` resources for those controls not created programmatically.

typedef

The data type `docStruc` is a structure comprising fields in which the references to the control objects for the various controls will be stored. A handle to this structure will be stored in the window's window object.

Global Variables

`actionFunctionVertUPP` and `actionFunctionHorizUPP` will be assigned universal procedure pointers to action functions for the scroll bars.

main

Universal procedure pointers are created for the action functions for the two scroll bars.

The call to the function `copyPString` causes the string in the first parameter to be copied to the global variable `gCurrentString`. The string in `gCurrentString`, which will be changed at various points in the code, will be drawn in the window header frame.

The next block opens a window, makes the window's graphics port the current port, and sets the graphics port's font to the small system font. This latter is because one of the pop-up menus will use the window font. `SetThemeWindowBackground` is called to set the background color/pattern for the window. The window's background will be similar to that applying to Mac OS 8/9 dialogs, which is appropriate for a window containing nothing but controls.

The call to `NewHandle` gets a `relocatable` block the size of one `docStruc` structure. The handle to the block is stored in the window's window object by the call to SetWRefCon.

In the next block, `doGetControls` creates and draws the controls, `doAdjustScrollBars` resizes and locates the scroll bars, and sets their maximum value, according to the dimensions of the window's port rectangle, and `ShowWindow` makes the window visible.

Note that error handling here and in other areas of this demonstration program is somewhat rudimentary. In the unlikely event that certain calls fail, `ExitToShell` is called to terminate the program.

doGetControls

The function `doGetControls` creates the controls.

At the first line, if the program is running on Mac OS 8/9, `CreateRootControl` is called to create a root control for the window. On Mac OS 8/9, the first control created must be always be the root control (which is implemented as a user pane). This call is not necessary on Mac OS X because, on Mac OS X, root controls are created automatically for windows which have at least one control.

A handle to the structure in which the references to the control objects will be stored is then retrieved.

The controls are then created, some from `'CNTL'` resources using `GetNewControl` and some programmatically. All of these calls create a control object for the relevant control and insert the object into the control list for the specified window. `GetNewControl` draws the controls created from `'CNTL'` resources. In the case of controls created

programmatically, `ShowControl` must be called to cause the control to be drawn. The reference to each control object is assigned to the appropriate field of the window's "document" structure.

Because of the sequence in which the controls are created and initially drawn, the group boxes would ordinarily over-draw the radio buttons and checkboxes. However, the calls to `AutoEmbedControl` embed these latter controls in their respective group boxes, ensuring that they will be drawn after (or "on top of") the group boxes. (`AutoEmbedControl`, rather than `EmbedControl`, is used in this instance because the radio button rectangles are visually contained by their respective group box rectangles.)

The call to `SetControlData`, with `kControlPushButtonDefaultTag` passed in the third parameter, causes the default outline to be drawn around the specified push button.

In the next block, the title fonts of the left color icon variant and right color icon variant push buttons are changed. Firstly, the flags and font fields of a control font style structure are assigned constants so that the following call to `SetControlFontStyle` will set the title font of the left color icon variant push button to the small system font. The font field is then changed so that the second call to `SetControlFontStyle` will set the title font of the right color icon variant push button to the small emphasized system font.

Lastly, the checkbox titled Rulers is disabled. This is for the purpose of the latency aspect of the demonstration.

doMouseDown

`doMouseDown` switches according to the window part in which a `mouseDown` event occurs.

At the `inContent` case, if the window in which the mouse-down occurred is the front window, and since all of the controls are located in the window's content region, a call to the function `doInContent` is made.

The `inGrow` case is of particular significance to the scroll bars. Following the call to `ResizeWindow`, the function `doAdjustScrollBars` is called to erase, move, resize, and redraw the scroll bars and reset the control's maximum value according to the new size of the window. (The call to `doDrawMessage` is incidental to the demonstration. It simply redraws the window header frame and text in the window.)

The `inZoomIn`/`InZoomOut` case is also of significance to the scroll bars. If the call to `TrackBox` returns a non-zero value, the window's port rectangle is erased before `ZoomWindowIdeal` zooms the window. Following the call to `ZoomWindowIdeal` the function `doAdjustScrollBars` is called to hide, move, resize, and redraw the scroll bars.

doMenuChoice

`doMenuChoice` handles user choices from the pull-down menus.

The `mDemonstration` case handles the Demonstration menu. Reference to that menu and a handle to the window's "document" structure are obtained.

If the menu item is the Color item, `IsControlVisible` is called to determine the current visibility status of the Color group box. If it is visible, the call to `HideControl` hides the group box and its embedded radio buttons; also, the menu item text is changed to "Show Color." If it is not visible, `ShowControl` is called and the menu item text is changed to "Hide Color."

If the menu item is the Grids item, the same general sequence takes place in respect of the Grids group box. This time, however, `IsControlActive` is used to determine whether the control is active or inactive, and `ActivateControl` and `DeactivateControl` are called, and the menu item is toggled, as appropriate. Note that, because of latency, the application does not have to "remember" that one of the embedded checkboxes was deactivated at program start. The Control Manager does the remembering.

doUpdate

`doUpdate` is called whenever the application receives an update event for its window. Between the usual calls to `BeginUpdate` and `EndUpdate` (ignored on Mac OS X), the window's graphics port is set as the current port for drawing, and `UpdateControls` is called to draw those controls intersecting the current visible region (which, between the `BeginUpdate` and `EndUpdate` calls, equates to the Mac OS 8/9 update region). The line preceding the if block is incidental to the demonstration. It simply redraws the window header frame and text in the window.

doActivateWindow

`doActivateWindow` switches according to whether the specified window is becoming active or is about to be made inactive. (Actually, `doActivateWindow` will never be called by `doActivate` in this program because the program only opens one window. It will however, be called by the function `doOSEvent`.)

At the first line, `GetRootControl` gets a reference to the window's root control.

If the window is becoming active, `ActivateControl` is called to activate the root control. Since all other controls are embedded in the root control, all controls will be activated by this call.

If the window is about to become inactive, `DeactivateControl` is called to deactivate the root control. Since all other controls are embedded in the root control, all controls will be deactivated by this call.

The calls to `doDrawMessage` are incidental to the demonstration. They simply redraw the window header frame and text in the window in the appropriate mode (inactive or active).

doOSEvent

`doOSEvent` handles operating system events. If the event is a suspend or resume event, a global variable is then set to indicate whether the program has come to the foreground or has been sent to the background. This global variable is used in `doUpdate`, and controls the color in which the text in the window header is drawn.

doInContent

`doInContent` further processes mouse-down events in the content region. Because the content region of the window contains nothing but controls, this function is really just the main switching point for the further handling of those controls.

The first line gets the handle to the "document" structure containing the references to the various control objects. The call to `SetPortWindowPort` is a necessary precursor to the call to `GlobalToLocal`, which converts the mouse coordinates in the event structure's where field from global coordinates to the local coordinates required in the following call to `FindControl`. (`FindControl` is used here rather than the newer function `FindControlUnderMouse` because there is no requirement to get a reference to a control even if no part was hit and no requirement to determine whether a mouse-down event has occurred in a deactivated control.)

If there is a control at the cursor location at which the mouse button is released, the control reference returned by the `FindControl` call is first compared with the references to the pop-up menu controls stored in the window's "document" structure. If a match is found, `TrackControl` is called with (`ControlActionUPP`) -1 passed in the `actionProc` parameter so as to cause an action function within the control's CDEF to be repeatedly invoked while the mouse button remains down. When `TrackControl` returns, the control value is obtained by a call to `GetControlValue` and a function is called to perform further handling

Note that `TrackControl`, rather than the newer function `HandleControlClick`, is used in this program because none of the controls require modifier keys to be passed in. (Of course, `HandleControlClick` would work just as well (with 0 passed in the `inModifiers` parameter).)

If the control reference returned by `FindControl` does not match the pop-up controls' references, it is then tested against the references to the vertical and horizontal scroll bar controls. If it matches the reference to the vertical scroll bar (which uses the non-live-feedback CDEF variant), the function `doVertScrollbar` is called to perform further handling. If it matches the reference to the horizontal scroll bar (which uses the live-feedback CDEF variant), `TrackControl` is called (`ControlActionUPP`) -1 passed in the `actionProc` parameter. This latter is because the UPP to the control action function has already been set. (Recall that it was passed in a parameter of the `CreateScrollBarControl` call which created the control.) The net effect of is that the application-defined action function to which the UPP relates will be repeatedly called while the mouse button remains down.

If the reference returned by `FindControl` does not match the references to any of the pop-up menu buttons or scroll bars, it must be a reference to one of the other controls. In this case, `TrackControl` is called, with the `procPtr` field set to that required for push buttons, radio buttons, and checkboxes (that is, NULL). If the cursor is still within the control when the mouse button is released, the reference is compared to, in sequence, the references to the radio buttons, the checkboxes, and the push buttons. If a match is found, the appropriate function is called to perform further handling.

doPopupMenuChoice

`doPopupMenuChoice` further handles mouse-downs in the pop-up menu buttons. In this demonstration, the further handling that would normally take place here is replaced by simply drawing the menu item text in the window.

The call to `GetControlData` gets a reference to the control's menu, allowing `GetMenuItemText` to get the item text into a global variable. This allows the text to be drawn in the window header frame.

doVertScrollbar

`doVertScrollbar` is called from `doInContent` in the case of a mouse-down in the vertical scroll bar (which uses the non-live-feedback variant of the CDEF).

The call to `copyPString` is incidental to the demonstration. It simply copies some appropriate text to the global variable `gCurrentString`.

At the next line, the function switches on the control part code. If the control part code was the scroll box/scroller (that is, the "indicator"), `TrackControl` is called with the `procPtr` parameter set to that required for the scroll box of non-live-feedback scroll bars (that is, NULL). If the user did not move the cursor outside the control before releasing it, the if block executes, retrieving the new control value, converting it to a string, appending that string to the string currently in `gCurrentString`, and drawing

`gCurrentString` in the window header. (In a real application, calculation of the distance and direction to scroll, and the scrolling itself, would take place inside this if block.)

If the mouse down was in the gray area/track or one of the scroll arrows, `TrackControl` is called with a Universal Procedure Pointer (UPP) passed in the `actionProc` parameter. The effect of this is that the application-defined action function to which the UPP relates will be repeatedly called while the mouse button remains down.

Action Functions

An action function is an example of a "callback function" (sometimes called a "hook function"). A callback function is an application-defined function that is called by a `Toolbox` function during the `Toolbox` function's execution, thus extending the features of the `Toolbox` function.

actionFunctionVert

`actionFunctionVert` is the action function called by `TrackControl` at the bottom of `doVertScrollbar`. Because it is repeatedly called by `TrackControl` while the mouse button remains down, the scrolling such a function would perform in a real application continues repeatedly until the mouse button is released (provided the cursor remains within the scroll arrow or gray area/track).

The call to `copyPString` is incidental to the demonstration. It simply copies some appropriate text to the global variable `gCurrentString`.

The `if(controlPartCode)` line ensures that, if the cursor is not still inside the scroll arrow or gray area/track, the action function exits and all scrolling ceases until the user brings the cursor back within the scroll arrow or gray area/track, causing a non-zero control part code to be again received. The following occurs only when the cursor is within the control.

The function switches on the control part code. If the mouse-down is in a scroll arrow, the variable `scrollDistance` is set to 2. If it is in a gray area, `scrollDistance` is set to 55. (In this simple demonstration, these are just arbitrary values. In a real application, you would assign an appropriate value in the case of the scroll arrows, and assign a calculated value (based primarily on current window height) in the case of the gray areas/tracks.)

The next block converts the value in `scrollDistance` to the required negative value if the user is scrolling down rather than up.

The next block defeats any further scrolling action if the down scroll arrow is being used and the "document" is at the maximum scrolled position or the up scroll arrow is being used and the "document" is at the minimum scrolled position.

The distance to scroll having been set, the call to the function `doMoveScrollBox` moves the scroll box/scroller the appropriate distance in the appropriate direction and updates the control's value accordingly. This means, of course, that the scroll box/scroller is being continually moved, and the control's value continually updated, while the mouse button remains down.

In this demonstration, the remaining action is to retrieve the current value of the control, convert it to a string, append it to the string currently in `gCurrentString`, and draw `gCurrentString` in the window header frame. (In a real application, the actual scrolling of the window's contents would be affected here.)

actionFunctionHoriz

`actionFunctionHoriz` is the action function passed in the `actionProc` parameter of the `TrackControl` call in `doInContent` arising from a mouse-down in the horizontal scroll bar. This action function differs from that for the vertical scroll bar because the horizontal scroll bar uses the live-feedback variant of the CDEF.

The principal differences are that action functions for live-feedback scroll bars must continually scroll the window's contents, not only while the mouse button remains down in the scroll arrows and gray areas, but also while the scroll box/scroller is being dragged. Accordingly, this function, unlike the action function for the vertical scroll bar, is also called while the mouse button remains down in the scroll box/scroller.

The call to `copyPString` is incidental to the demonstration. It simply copies some appropriate text to the global variable `gCurrentString`.

If the mouse-down occurred in the scroll box/scroller, the code which sets up the scroll distance, adjusts the sign of the scroll distance according to whether the scroll is left or right, prevents scrolling beyond the minimum and maximum scroll values, and calls `doMoveScrollBox` to move the scroll box/scroller and update the control's value, is bypassed. The call to `doMoveScrollBox` is bypassed because, in live-feedback, the CDEF moves the scroll box/scroller and updates the control's value when the mouse-down is in the scroll box/scroller.

In this demonstration, the action taken after the main block of code has been bypassed (mouse-down in the scroll box/scroller) or executed (mouse-down in the scroll arrows or gray area/track) is to retrieve the current value of the control, convert it to a string, append it to the string currently in `gCurrentString`, and draw `gCurrentString` in the window header frame. (In a real application, the actual scrolling of the window's contents would be affected here.)

doMoveScrollBox

`doMoveScrollBox` is called from within the action functions to reset the control's current value to reflect the scrolled distance, and to reposition the scroll box/scroller accordingly.

The first two lines retrieve the control's current value and maximum value. The next line calculates the new control value by subtracting the distance to scroll received from the calling action function from the current control value. The next four lines prevent the control's value from being set lower or higher than the control's minimum and maximum values respectively. The call to `SetControlValue` sets the new control value and repositions the scroll box/scroller.

doRadioButtons

`doRadioButtons` is called when the mouse-down is within a radio button. The first three calls to `SetControlValue` set all radio buttons to the off state. The final call sets the radio button under the mouse to the on state.

doCheckboxes

`doCheckboxes` is called when the mouse-down is within a checkbox. The single line simply toggles the current value of the control.

doPushButtons

`doPushButtons` is called when the mouse-down is within a push button. In this demonstration, the only action taken is to draw the identity of the push button in the window header frame.

doAdjustScrollBars

`doAdjustScrollBars` is called if the user resizes or zooms the window.

At the first line, a handle to the window's "document" structure is retrieved from the window object.

At the next line, the coordinates representing the window's current content region are assigned to a Rect variable which will be used in calls to MoveControl and SizeControl.

`MoveControl` and `SizeControl` both redraw the specified scroll bar. Because `SizeControl` will be called immediately after `MoveControl`, this will cause a very

slight flickering of the scroll bars. To prevent this, the scroll bars will be hidden while these two functions are executing.

The calls to `HideControl` hide the scroll bars. The calls to `MoveControl` erase the scroll bars, offset the `contrlRect` fields of their control structures, and redraw the scroll bars within the offset rectangle. `SizeControl` hides the scroll bars (in this program they are already hidden), adjusts the `contrlRect` fields of their control structures, and redraws the scroll bars within their new rectangles. The calls to `ShowControl` then show the scroll bars.

In this demonstration, the remaining lines set the new maximum values for the scroll bars according to the new height and width of the window. No attempt is made to calculate the required new control value to ensure that the (non-existent) document remains in the same scrolled position after the zoom or resize. In a real application, this, plus the calculation of the maximum value according to, for example, the line height of text content as well as the new window height, are matters that would need to be attended to in this function.

Demonstration Program Controls2 Listing

```
//
**********************************************************************************************
// Controls2.c                                                          CLASSIC EVENT
MODEL
//
**********************************************************************************************
//
// This program:
//
// •  Opens a kWindowDocumentProc window with a two horizontal scroll bars, each of which
//    relates to the picture displayed immediately above it.
//
// •  Allows the user to horizontally scroll the pictures within the window using the scroll
//    box/scroller, the scroll arrows and the gray area/track of each scroll bar.
//
// The top scroll bar uses the non-live-feedback variant of the scroll bar CDEF. The bottom
// scroll bar uses the live-feedback variant.
//
// With regard to the scroll bars, the principal differences between this program and
// Controls1 are that, in this program:
//
// •  The scroll bar scroll boxes are made proportional.
//
// •  The action functions are set using the function SetControlAction.
//
// •  References to the scroll bar controls are not stored in, and retrieved from, a
document
//    structure associated with the window. Instead,  each control is assigned a controlID
//    using SetControlID, allowing the ID of the control to be retrieved using GetControlID
//    and a reference to the control to be obtained using GetControlByID.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  An 'MBAR' resource, and 'MENU' resources for Apple, File and Edit (preload, non-
//    purgeable).
//
// •  A 'WIND' resource (purgeable) (initially visible).
//
// •  Two 'CNTL' resource for the horizontal scroll bars (purgeable).
//
// •  'PICT' resources containing the pictures to be scrolled (non-purgeable).
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
//
**********************************************************************************************
// ..............................................................................
includes

#include <Carbon.h>

// ..............................................................................
defines

#define rMenubar            128
#define rNewWindow          128
#define rPictureNonLive     128
```

```
#define rPictureLive        129
#define mAppleApplication   128
#define  iAbout             1
#define mFile               129
#define  iQuit              12
#define cScrollbarNonLive   128
#define cScrollbarLive      129
#define kScrollbarNonLiveID 1
#define kScrollbarLiveID    2
#define MAX_UINT32          0xFFFFFFFF

// ..................................................................... global
variables

ControlActionUPP gActionFuncNonLiveUPP;
ControlActionUPP gActionFuncLiveUPP;
Boolean          gDone;
Rect             gPictRectNonLive, gPictRectLive;
PicHandle        gPictHandleNonLive, gPictHandleLive ;

// ..................................................................... function
prototypes

void  main               (void);
void  doPreliminaries    (void);
OSErr quitAppEventHandler (AppleEvent *,AppleEvent *,SInt32);
void  doEvents           (EventRecord *);
void  doMouseDown        (EventRecord *);
void  doUpdate           (EventRecord *);
void  doActivate         (EventRecord *);
void  doActivateWindow   (WindowRef,Boolean);
void  doOSEvent          (EventRecord *);
void  doMenuChoice       (SInt32);
void  doInContent        (EventRecord *,WindowRef);
void  doNonLiveScrollBars (ControlPartCode,WindowRef,ControlRef,Point);
void  actionFuncNonLive  (ControlRef,ControlPartCode);
void  actionFuncLive     (ControlRef,ControlPartCode);
void  doMoveScrollBox    (ControlRef,SInt16);

// ************************************************************************************
main

void  main(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  WindowRef     windowRef;
  ControlRef    controlRefScrollbarNonLive, controlRefScrollbarLive;
  ControlID     controlID;
  Rect          portRect;
  EventRecord   eventStructure;

  // ..................................................................... do
preliminaries

  doPreliminaries();

  // ................................................. create universal procedure
pointers

  gActionFuncNonLiveUPP = NewControlActionUPP((ControlActionProcPtr) actionFuncNonLive);
  gActionFuncLiveUPP    = NewControlActionUPP((ControlActionProcPtr) actionFuncLive);

  // ..................................................... set up menu bar and
menus
```

```
  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    ExitToShell();
  SetMenuBar(menubarHdl);
  DrawMenuBar();

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
      DisableMenuItem(menuRef,0);
    }
  }

  // ..................................................................... open a
window

  if(!(windowRef = GetNewCWindow(rNewWindow,NULL,(WindowRef)-1)))
    ExitToShell();

  SetPortWindowPort(windowRef);

  // ................................... get controls and set ID and control action
functions

  controlRefScrollbarNonLive  = GetNewControl(cScrollbarNonLive,windowRef);
  controlID.signature = 'kjB ';
  controlID.id = kScrollbarNonLiveID;
  SetControlID(controlRefScrollbarNonLive,&controlID);

  SetControlAction(controlRefScrollbarNonLive,gActionFuncNonLiveUPP);

  controlRefScrollbarLive = GetNewControl(cScrollbarLive,windowRef);
  controlID.id = kScrollbarLiveID;
  SetControlID(controlRefScrollbarLive,&controlID);

  SetControlAction(controlRefScrollbarLive,gActionFuncLiveUPP);
  // ..................................................................... get
picture

  if(!(gPictHandleNonLive = GetPicture(rPictureNonLive)))
    ExitToShell();
  gPictRectNonLive = (*gPictHandleNonLive)->picFrame;

  if(!(gPictHandleLive = GetPicture(rPictureLive)))
    ExitToShell();
  gPictRectLive = (*gPictHandleLive)->picFrame;
  OffsetRect(&gPictRectLive,0,191);

  // ..................................................... set up for proportional scroll
boxes

  GetWindowPortBounds(windowRef,&portRect);
  SetControlViewSize(controlRefScrollbarNonLive,portRect.right);
  SetControlViewSize(controlRefScrollbarLive,portRect.right);

  // ..................................................................... enter
eventLoop

  gDone = false;
```

```
  while(!gDone)
  {
    if(WaitNextEvent(everyEvent,&eventStructure,MAX_UINT32,NULL))
      doEvents(&eventStructure);
  }
}

// **************************************************************************
doPreliminaries

void  doPreliminaries(void)
{
  OSErr osError;

  MoreMasterPointers(32);
  InitCursor();
  FlushEvents(everyEvent,0);

  osError = AEInstallEventHandler(kCoreEventClass,kAEQuitApplication,
                            NewAEEventHandlerUPP((AEEventHandlerProcPtr)
quitAppEventHandler),
                            0L,false);
  if(osError != noErr)
    ExitToShell();
}

// **************************************************************************
doQuitAppEvent

OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  OSErr    osError;
  DescType returnedType;
  Size     actualSize;
  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,
NULL,0,
                              &actualSize);

  if(osError == errAEDescNotFound)
  {
    gDone = true;
    osError = noErr;
  }
  else if(osError == noErr)
    osError = errAEParamMissed;

  return osError;
}

// **************************************************************************
doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
  switch(eventStrucPtr->what)
  {
    case kHighLevelEvent:
      AEProcessAppleEvent(eventStrucPtr);
      break;

    case mouseDown:
      doMouseDown(eventStrucPtr);
      break;

    case updateEvt:
      doUpdate(eventStrucPtr);
```

```
      break;

    case activateEvt:
      doActivate(eventStrucPtr);
      break;

    case osEvt:
      doOSEvent(eventStrucPtr);
      break;
  }
}

// ****************************************************************************
doMouseDown

void  doMouseDown(EventRecord *eventStrucPtr)
{
  WindowRef      windowRef;
  WindowPartCode partCode;

  partCode = FindWindow(eventStrucPtr->where,&windowRef);

  switch(partCode)
  {
    case inMenuBar:
      doMenuChoice(MenuSelect(eventStrucPtr->where));
      break;

    case inContent:
      if(windowRef != FrontWindow())
        SelectWindow(windowRef);
      else
        doInContent(eventStrucPtr,windowRef);
      break;

    case inDrag:
      DragWindow(windowRef,eventStrucPtr->where,NULL);
      break;
  }
}

// ****************************************************************************
doUpdate

void  doUpdate(EventRecord *eventStrucPtr)
{
  WindowRef  windowRef;
  RgnHandle  regionHdl;
  ControlID  controlID;
  ControlRef controlRef;

  windowRef = (WindowRef) eventStrucPtr->message;

  BeginUpdate(windowRef);

  SetPortWindowPort(windowRef);

  regionHdl = NewRgn();
  if(regionHdl)
  {
    GetPortVisibleRegion(GetWindowPort(windowRef),regionHdl);
    UpdateControls(windowRef,regionHdl);
    DisposeRgn(regionHdl);
  }

  controlID.signature = 'kjB ';
```

```
  controlID.id = kScrollbarNonLiveID;
  GetControlByID(windowRef,&controlID,&controlRef);
  SetOrigin(GetControlValue(controlRef),0);
  DrawPicture(gPictHandleNonLive,&gPictRectNonLive);
  SetOrigin(0,0);

  controlID.id = kScrollbarLiveID;
  GetControlByID(windowRef,&controlID,&controlRef);
  SetOrigin(GetControlValue(controlRef),0);
  DrawPicture(gPictHandleLive,&gPictRectLive);
  SetOrigin(0,0);
  EndUpdate(windowRef);
}

// ****************************************************************************
doActivate

void  doActivate(EventRecord *eventStrucPtr)
{

  WindowRef windowRef;
  Boolean   becomingActive;

  windowRef = (WindowRef) eventStrucPtr->message;
  becomingActive = ((eventStrucPtr->modifiers & activeFlag) == activeFlag);
  doActivateWindow(windowRef,becomingActive);
}

// ****************************************************************************
doActivateWindow

void  doActivateWindow(WindowRef windowRef,Boolean becomingActive)
{
  ControlID  controlID;
  ControlRef controlRefScrollbarNonLive, controlRefScrollbarLive;

  controlID.signature = 'kjB ';
  controlID.id = kScrollbarNonLiveID;
  GetControlByID(windowRef,&controlID,&controlRefScrollbarNonLive);
  controlID.id = kScrollbarLiveID;
  GetControlByID(windowRef,&controlID,&controlRefScrollbarLive);

  if(becomingActive)
  {
    ActivateControl(controlRefScrollbarNonLive);
    ActivateControl(controlRefScrollbarLive);
  }
  else
  {
    DeactivateControl(controlRefScrollbarNonLive);
    DeactivateControl(controlRefScrollbarLive);
  }
}

// ****************************************************************************
doOSEvent

void  doOSEvent(EventRecord *eventStrucPtr)
{
  switch((eventStrucPtr->message >> 24) & 0x000000FF)
  {
    case suspendResumeMessage:
      if((eventStrucPtr->message & resumeFlag) == 1)
        SetThemeCursor(kThemeArrowCursor);
      break;
  }
```

```
}
// ****************************************************************************
doMenuChoice

void  doMenuChoice(SInt32 menuChoice)
{
  MenuID        menuID;
  MenuItemIndex menuItem;

  menuID = HiWord(menuChoice);
  menuItem = LoWord(menuChoice);

  if(menuID == 0)
    return;

  switch(menuID)
  {
    case mAppleApplication:
      if(menuItem == iAbout)
        SysBeep(10);
      break;

    case mFile:
      if(menuItem == iQuit)
        gDone = true;
      break;
  }

  HiliteMenu(0);
}

// ****************************************************************************
doIncontent

void  doInContent(EventRecord *eventStrucPtr,WindowRef windowRef)
{
  ControlPartCode controlPartCode;
  ControlRef      controlRef;
  ControlID       controlID;

  SetPortWindowPort(windowRef);
  GlobalToLocal(&eventStrucPtr->where);

  if(controlPartCode = FindControl(eventStrucPtr->where,windowRef,&controlRef))
  {
    GetControlID(controlRef,&controlID);

    if(controlID.id == kScrollbarNonLiveID)
      doNonLiveScrollBars(controlPartCode,windowRef,controlRef,eventStrucPtr->where);
    else if(controlID.id == kScrollbarLiveID)
      TrackControl(controlRef,eventStrucPtr->where,(ControlActionUPP) -1);
  }
}
// ****************************************************************************
doNonLiveScrollBars

void  doNonLiveScrollBars(ControlPartCode controlPartCode,WindowRef windowRef,
                          ControlRef controlRef,Point mouseXY)
{
  SInt16    oldControlValue;
  SInt16    scrollDistance;
  RgnHandle updateRgnHdl;

  switch(controlPartCode)
  {
    case kControlIndicatorPart:
```

```
      oldControlValue = GetControlValue(controlRef);
      if(TrackControl(controlRef,mouseXY,NULL))
      {
        scrollDistance = oldControlValue - GetControlValue(controlRef);
        if(scrollDistance != 0)
        {
          updateRgnHdl = NewRgn();
          ScrollRect(&gPictRectNonLive,scrollDistance,0,updateRgnHdl);
          InvalWindowRgn(windowRef,updateRgnHdl);
          DisposeRgn(updateRgnHdl);
        }
      }
      break;

    case kControlUpButtonPart:
    case kControlDownButtonPart:
    case kControlPageUpPart:
    case kControlPageDownPart:
      TrackControl(controlRef,mouseXY,(ControlActionUPP) -1);
      break;
  }
}

// ********************************************************************
actionFuncNonLive

void  actionFuncNonLive(ControlRef controlRef,ControlPartCode controlPartCode)
{
  WindowRef windowRef;
  SInt16    scrollDistance, controlValue;
  Rect      portRect;
  RgnHandle updateRgnHdl;

  if(controlPartCode)
  {
    windowRef = GetControlOwner(controlRef);

    switch(controlPartCode)
    {
      case kControlUpButtonPart:
      case kControlDownButtonPart:
        scrollDistance = 2;
        break;
      case kControlPageUpPart:
      case kControlPageDownPart:
        GetWindowPortBounds(windowRef,&portRect);
        scrollDistance = (portRect.right - portRect.left - 10);
        break;
    }

    if((controlPartCode == kControlDownButtonPart) ||
       (controlPartCode == kControlPageDownPart))
      scrollDistance = -scrollDistance;

    controlValue = GetControlValue(controlRef);
    if(((controlValue == GetControlMaximum(controlRef)) && scrollDistance < 0) ||
       ((controlValue == GetControlMinimum(controlRef)) && scrollDistance > 0))
      return;

    doMoveScrollBox(controlRef,scrollDistance);

    if(controlPartCode == kControlUpButtonPart ||
       controlPartCode == kControlDownButtonPart)
    {
      updateRgnHdl = NewRgn();
      ScrollRect(&gPictRectNonLive,scrollDistance,0,updateRgnHdl);
```

```
      InvalWindowRgn(windowRef,updateRgnHdl);
      DisposeRgn(updateRgnHdl);
      BeginUpdate(windowRef);
    }

    SetOrigin(GetControlValue(controlRef),0);
    DrawPicture(gPictHandleNonLive,&gPictRectNonLive);
    SetOrigin(0,0);

    if(controlPartCode == kControlUpButtonPart || controlPartCode == kControlDownButtonPart)
      EndUpdate(windowRef);
  }
}

// ************************************************************************
actionFuncLive

void  actionFuncLive(ControlRef controlRef,ControlPartCode partCode)
{
  WindowRef windowRef;
  SInt16    scrollDistance, controlValue;
  Rect      portRect;

  windowRef = GetControlOwner(controlRef);

  if(partCode != 0)
  {
    if(partCode != kControlIndicatorPart)
    {
      switch(partCode)
      {
        case kControlUpButtonPart:
        case kControlDownButtonPart:
          scrollDistance = 2;
          break;

        case kControlPageUpPart:
        case kControlPageDownPart:
          GetWindowPortBounds(windowRef,&portRect);
          scrollDistance = (portRect.right - portRect.left) - 10;
          break;
      }

      if((partCode == kControlDownButtonPart) || (partCode == kControlPageDownPart))
        scrollDistance = -scrollDistance;

      controlValue = GetControlValue(controlRef);
      if(((controlValue == GetControlMaximum(controlRef)) && scrollDistance < 0) ||
         ((controlValue == GetControlMinimum(controlRef)) && scrollDistance > 0))
        return;

      doMoveScrollBox(controlRef,scrollDistance);
    }

    SetOrigin(GetControlValue(controlRef),0);
    DrawPicture(gPictHandleLive,&gPictRectLive);
    SetOrigin(0,0);
  }
}

// ************************************************************************
doMoveScrollBox

void doMoveScrollBox(ControlRef controlRef,SInt16 scrollDistance)
{
  SInt16 oldControlValue, controlValue, controlMax;
```

```
  oldControlValue = GetControlValue(controlRef);
  controlMax = GetControlMaximum(controlRef);

  controlValue = oldControlValue - scrollDistance;

  if(controlValue < 0)
    controlValue = 0;
  else if(controlValue > controlMax)
    controlValue = controlMax;

  SetControlValue(controlRef,controlValue);
}

//
****************************************************************************************************
```

Demonstration Program Controls2 Comments

This program is basically an extension of the scroll bars' aspects of the demonstration program Controls1 in that, unlike the scroll bars in Controls1, the scroll bars in this program actually scroll the contents of the window. Also, this program supports proportional scroll boxes/scrollers.

When the program is run, the user should scroll the pictures by dragging the scroll boxes/ scrollers, clicking in the scroll bar gray areas/tracks, clicking in the scroll arrows and holding the mouse button down while the cursor is in the gray areas/tracks and scroll arrows. The user should note, when scrolling with the scroll boxes/scrollers, that the top scroll bar uses the non-live-feedback variant of the scroll bar CDEF and the bottom scroll bar uses the live-feedback variant, this latter to facilitate the program's live-scrolling of the bottom picture.

The pictures scrolled in this demonstration are, respectively 1220 by 175 pixels and 915 by 175 pixels. "pane" for each picture and scroll bar is 400 pixels wide by 175 pixels high, the 'CNTL' resources set the control maximum values to 820 and 515 respectively, and the control rectangles specified in the 'CNTL' resource locate the scroll bars in the correct position in the non-resizable, non-zoomable window.

As an incidental aspect of the demonstration, two different methods are used to scroll the pictures when the scroll arrows are being used. In the top picture, at each pass through the action function, the pixels are scrolled using ScrollRect, the "vacated"area is invalidated, and only this vacated area is redrawn. In the bottom picture, at each pass through the action function, the whole visible part of the picture is redrawn. The user should note that the first method results in some flickering in the "vacated" area when the picture is scrolled, and that the second method eliminates this flickering at the cost of some horizontal "tearing" of the picture caused by the way in which the image is drawn by the monitor on its screen.

The following comments are limited to those areas which are significantly different from the same areas in the demonstration program Controls1.

defines

`kScrollbarNonLiveID` and `kScrollbarLiveID` will be assigned as the IDs of, respectively, the top (non-live) scroll bar and the bottom (live) scroll bar.

main

Two calls to `GetNewControl` allocate memory for the control objects, insert the control

objects into the window's control list and draw the controls.

Following each call to `GetNewControl`

- The id field of a variable of type `ControlID` is assigned the appropriate ID value for the specific control and the ID is then assigned to the control by a call to `SetControlID`.
- `SetControlAction` is called with a UPP passed in the `actionProc` parameter. The effect of this is that the application-defined action function to which the UPP relates will be repeatedly called while the mouse button remains down. As a consequence of using `SetControlAction`, (`ControlActionUPP`) -1 will be passed in `TrackControl`'s `actionProc` parameter.

Note that no root control is created in this program; accordingly, the two controls will be activated and deactivated individually.

In the next block, two `'PICT'` resources are loaded, the associated handles being assigned to two global variables. In each case, the picture structure's `picFrame` field (a `Rect`) is copied to a global variable. In the case of the second picture, this rectangle is then offset downwards by 191 pixels. (Note that the two `'PICT'` resources were created so that the top and left fields of the `picFrame` `Rect` are both zero.)

In the next block, the width of the port rectangle is passed in the `newViewSize` parameter of calls to `SetControlViewSize`. (In this case, the view width is the same as the width of the port rectangle. This value is in the same units of measurement as are used for the scroll bar minimum, maximum, and current values.) This makes the scroll boxes proportional (on Mac OS 8/9, provided that the user has selected Smart Scrolling on in the Options tab of the Appearance control panel).

doUpdate

In the two blocks which draw the pictures, the first call to `SetOrigin` sets the window origin to the current scroll position, that is, to the position represented by the control's current value, thus ensuring that the correct part of the picture will be drawn by the call to `DrawPicture`. The second call to `SetOrigin` resets the window's origin to (0,0).

Note that `GetControlByID` is used to retrieve references to the two controls.

DoActivateWindow

Note that `GetControlByID` is used to retrieve references to the two controls.

doInContent

`doInContent` establishes whether a mouse-down event was in one of the scroll bars and, if so, branches accordingly.

The call to `GlobalToLocal` converts the global coordinates of the mouse-down, stored in the where field of the event structure, to the local coordinates required by `FindControl`. If the call to `FindControl` returns a non-zero result, the mouse-down was in a scroll bar.

If the mouse-down was in a scroll bar, `GetControlID` is called to get the ID of the control. Then, as in the demonstration program Controls1

- If the mouse-down was in the non-live-feedback scroll bar, one of the application's functions is called to further handle the mouse-down event.
- If the mouse-down was in the live-feedback scroll bar, `TrackControl` is called with (`ControlActionUPP`) -1 passed in the `actionProc` parameter. This means that the application-defined (callback) function associated with the UPP previously set by `SetControlAction` will be continually called while the mouse button remains down.

doNonLiveScrollBars

`doNonLiveScrollBars` is similar to its sister function in Controls1 except that it actually scrolls the window's contents.

At the first line, the function switches on the control part code:

- If the mouse-down was in the scroll box/scroller (that is, the "indicator"), the control's value at the time of the mouse-down is retrieved. Control is then handed over to `TrackControl`, which tracks user actions while the mouse button remains down. If the user releases the mouse button with the cursor inside the scroll box/scroller, the scroll distance (in pixels) is calculated by subtracting the control's value prior to the scroll from its current value. If the user moved the scroll box/scroller, the picture's pixels are scrolled by the specified scroll distance in the appropriate direction, and the "vacated" area of the window following the scroll is added to the (currently empty) window update region (Mac OS 8/9). This means that an update event will be generated for the window and that the redraw of the picture will be attended to in the `doUpdate` function.
- If the mouse-down was in a scroll arrow or gray area/track, more specifically in one of the non-live-feedback's scroll bar's scroll arrows or gray areas/tracks, `TrackControl` takes control until the user releases the mouse button. The third parameter in the TrackControl call means that the application-defined (callback) function associated with the UPP set by `SetControlAction` will be continually called while the mouse button remains down.

actionFunctionNonLive

`actionFunctionNonLive` is the action function for the non-live-feedback scroll bar. Because it is repeatedly called by `TrackControl` while the mouse button remains down, the scrolling it performs continues repeatedly until the mouse button is released.

If the cursor is not still inside the scroll arrow or gray area/track, the action function exits. The following occurs only when the cursor is within the control.

A reference to the window which "owns" this control is retrieved from the control object.

If the control part being used by the user to perform the scrolling is one of the scroll arrows, the distance to scroll (in pixels) is set to 2. If the control part being used is one of the gray areas/track, the distance to scroll is set to the width of the window's content region minus 10 pixels. (Subtracting 10 pixels ensures that a small part of the pre-scroll display will appear at right or left (depending on the direction of scroll) of the post-scroll display.)

The first block following the switch converts the distance to scroll to the required negative value if the user is scrolling towards the right. The second block defeats any further scrolling action if the left scroll arrow is being used, the mouse button is still down and the document is at the minimum (left) scrolled position or the right scroll arrow is being used, the mouse button is still down and the document is at the maximum (right) scrolled position.

With the scroll distance determined, the call to the function `doMoveScrollBox` adds/subtracts the distance to scroll to/from the control's current value and repositions the scroll box/scroller accordingly.

At this stage, the picture scrolling takes place. If scrolling is being effected using the scroll arrows, `ScrollRect` scrolls the picture's pixels by the specified amount, and in the specified direction, as represented by the distance-to-scroll value. The "vacated" area is then added to the window's update region (previously empty) by the call to `InvalWindowRgn`, and `BeginUpdate` is called to ensure that, on Mac OS 8/9, only the "vacated" area will be redrawn and the update region is cleared.

Regardless of whether the picture is being scrolled using the scroll arrows or the gray areas, `SetOrigin` is then called to reset the window origin so that that part of the picture represented by the current scroll position is drawn. After the correct part of the picture is drawn, the window origin is reset to (0,0).

Finally, if `BeginUpdate` was called prior to the draw (that is, scrolling is being effected using the scroll arrows), `EndUpdate` is called.

actionFunctionLive

`actionFunctionLive` is the action function for the live-feedback scroll bar.

The principal differences between this action function and the previous one are that action functions for live-feedback scroll bars must continually scroll the window's contents, not only while the mouse button remains down in the scroll arrows and gray areas/track, but also while the scroll box/scroller is being dragged. Accordingly, this action function, unlike the action function for the non-live-feedback scroll bar, is also called while the mouse button remains down in the scroll box/scroller.

If the mouse-down occurred in the scroll box/scroller, the code which sets up the scroll distance, adjusts the sign of the scroll distance according to whether the scroll is left or right, prevents scrolling beyond the minimum and maximum scroll values, and calls `doMoveScrollBox` to move the scroll box/scroller and update the control's value, is bypassed. The call to `doMoveScrollBox` is bypassed because the live-feedback variant of the CDEF moves the scroll box/scroller and updates the control's value when the mouse-down is in the scroll box/scroller.

After the if block has been bypassed (mouse-down in the scroll box/scroller) or executed (mouse-down in the scroll arrows or gray area/track), the window contents are scrolled. Regardless of whether the picture is being scrolled using the scroll box/scroller, the scroll arrows, or the gray areas/track, `SetOrigin` is called to reset the window origin so that that part of the picture represented by the current scroll position is drawn by the call to `DrawPicture`. After the correct part of the picture is drawn, the window origin is reset to (0,0).

Note that this alternative approach to redrawing the picture when scrolling is being effected using the scroll arrows has not been dictated by the fact that this is a live-feedback action function. Either of these two approaches will work in both live-feedback and non-live-feedback action functions.

doMoveScrollBox

`doMoveScrollBox` is called from within the action function to reset the control's current value to reflect the scrolled distance and to reposition the scroll box accordingly.

Chapter 8. Dialogs and Alerts

IN THIS CHAPTER

- Introduction
- Alert Types, Modalities, and Levels
- Dialog Types and Modalities
- Window Types for Alerts and Dialogs
- Content of Alerts and Dialogs
- Removing Dialogs
- Creating and Removing Alerts
- Creating Dialogs
- Items in Dialogs
- Displaying Alerts and Dialogs
- Handling Events in Alerts and Dialogs
- Closing and Disposing of Dialogs
- Creating Displaying and Handling Window-Modal (Sheet) Alerts and Dialogs
- Balloon Help for Dialogs-Mac OS 8/9
- Help Tags for Dialogs-Mac OS X
- Main Dialog Manager Constants, Data Types and Functions
- Relevant Window Manager Functions (Mac OS X Only)
- Demonstration Program DialogAndAlerts Listing
- Demonstration Program DialogsAndAlerts Comments

Introduction

Your application should present alerts to the user whenever an unexpected or undesirable situation occurs.

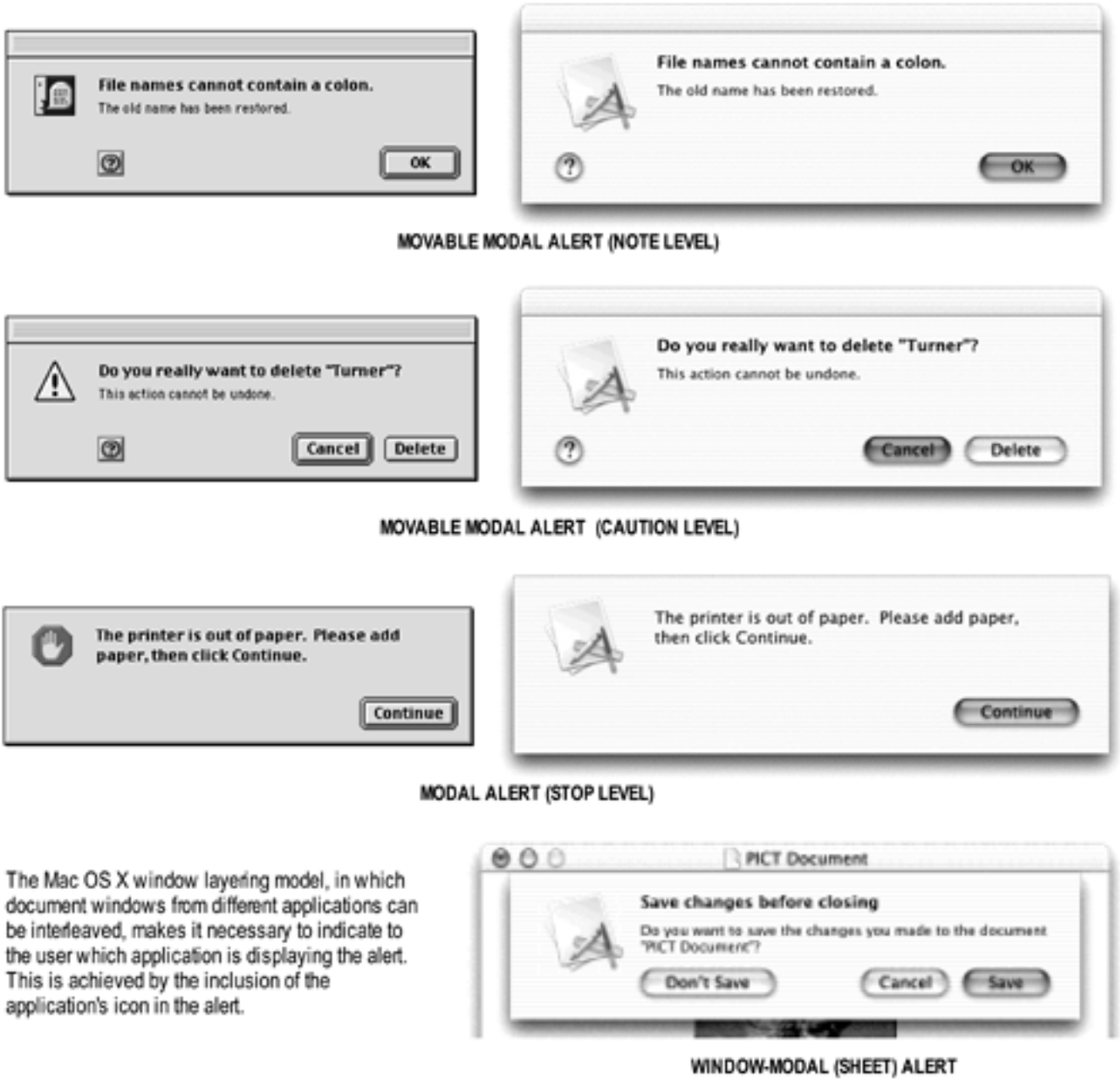
Before your application carries out a command, it may present a dialog to solicit additional information from the user or to allow the user to modify settings.

Alert Types, Modalities, and Levels

Alert Types and Modalities

There are three types of alert, namely, the **modal alert,** the **movable modal alert,** and, on Mac OS X only, the **sheet alert.** The three types are shown in Fig 1.

Figure 1. TYPES AND LEVELS OF ALERTS


File names cannot contain a colon.
The old name has been restored.
OK
File names cannot contain a colon.
The old name has been restored.
OK
MOVABLE MODAL ALERT (NOTE LEVEL)
Do you really want to delete "Turner"?
This action cannot be undone.
Cancel
Delete
Do you really want to delete "Turner"?
This action cannot be undone.
Cancel
Delete
MOVABLE MODAL ALERT (CAUTION LEVEL)
The printer is out of paper. Please add paper, then click Continue.
Continue
The printer is out of paper. Please add paper, then click Continue.
Continue
MODAL ALERT (STOP LEVEL)
The Mac OS X window layering model, in which document windows from different applications can be interleaved, makes it necessary to indicate to the user which application is displaying the alert. This is achieved by the inclusion of the application's icon in the alert.
PICT Document
Save changes before closing
Do you want to save the changes you made to the document "PICT Document"?
Don't Save
Cancel
Save
WINDOW-MODAL (SHEET) ALERT

Modal Alert

The fixed-position modal alert places the user in the state, or **mode,** of being able to work only inside the alert. The only response the user receives when clicking anywhere outside the alert is the alert sound. The modal alert is thus **system-modal,** meaning that is denies user interaction with anything but the alert until it is dismissed.

There will be very few, if any, situations where the use of a modal alert in your application is justified.

Movable Modal Alert

Movable modal alerts retain the essentially modal characteristic of their fixed-position counterpart, the main differences being that they allow the user to drag the alert so as to uncover obscured areas of an underlying window and bring another application to the front. Movable modal alerts are thus **application-modal.**

Window-Modal (Sheet) Alert-Mac OS X

Mac OS X introduced a new type of alert called the **sheet alert.** Sheet alerts, which are invariably attached to an owner window, are **window-modal.** The information conveyed by the alert, or the alternative actions requested, should pertain only to the document to whose window the alert is attached.

Levels of Alert

Modal and movable modal alerts can display one of three levels of alert depending on the nature of the situation the alert is reporting to the user (see Fig 1). The three levels of alert, which, on Mac OS 8/9, are identified by icons supplied automatically by the system, are as follows:

- ***Note Level.*** The note level is used to inform users of an occurrence that will not have serious consequences (see Fig 1). Usually, a note level alert simply offers information, although it may ask a simple question and provide, via the push buttons, a choice of responses.

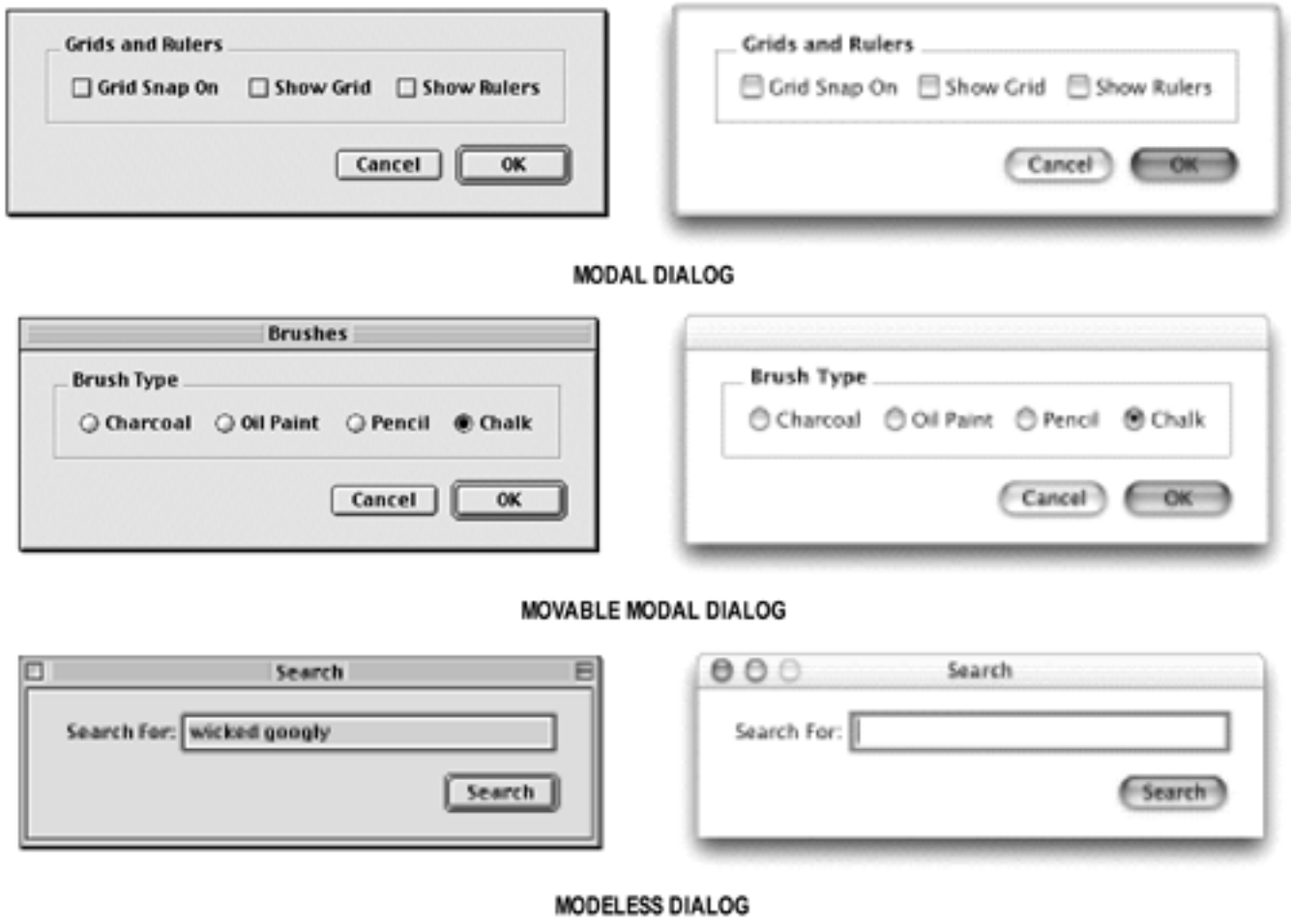
- ***Caution Level.*** The caution level is used to alert the user to an operation that may have undesirable results if it is allowed to continue. As shown in Fig 1, you should provide the user, via the push buttons, with a choice of whether to continue with or back out of the operation.

- ***Stop Level.*** The stop level is used to inform the user of a situation so serious that the operation cannot proceed.

Note that, at the time of writing, there was no visual distinction between alert levels on Mac OS X, the application icon rather than distinct note, caution, and stop icons being displayed. At the time of writing, it was expected that Carbon would eventually support the "badging" of the application icon with alert level badges similar to the Mac OS 8/9 note, caution, and stop icons.

Dialog Types and Modalities

There are four types of dialog, namely, modal dialogs, movable modal dialogs, modeless dialogs, and, on Mac OS X only, sheet dialogs. The four types are illustrated in the examples in Fig 2.



Modal Dialog

Fixed-position modal dialogs place the user in the state, or mode, of being able to work only inside the dialog. The only response the user receives when clicking outside the dialog is the alert sound. The modal alert is thus **system-modal,** meaning that it denies user interaction with anything but the dialog until it is dismissed.

There will be very few, if any, situations where the use of a modal dialog in your application is justified.

Movable Modal Dialog

Movable modal dialogs retain the essentially modal characteristic of their fixed-position counterpart, the main differences being that they allow the user to drag the dialog so as to uncover obscured areas of an underlying window and bring another application to the front. Movable modal dialogs are thus **application-modal.**

The absence of boxes/buttons in the title bar of a movable modal dialog visually indicates to the user that the dialog is modal rather than modeless.

Modeless Dialog

Modeless dialogs look like document windows, except for their interior color/pattern and, on Mac OS 8/9, a one-pixel frame just inside the window frame. Unlike document windows, however, modeless dialogs should not contain scroll bars or a size box/resize control.

Modeless dialogs should not require the user to dismiss them before the user is able to do anything else. Thus modeless dialogs should be made to behave much like document windows in that the user should be able to move them, bring other windows in front of them, and close them.

Modeless dialogs should ordinarily not have a Cancel push button, although they may have a Stop push button to halt long operations such as searching.

Window-Modal (Sheet) Dialog-Mac OS X

Mac OS X introduced a new type of dialog called the **sheet dialog.** Sheet dialogs, which are invariably attached to an owner window, are **window-modal.** The information or settings solicited by the dialog should pertain only to the document to whose window the dialog is attached.

Window Types for Alerts and Dialogs

Fig 3 shows the seven (eight on Mac OS X) available window types for alerts and dialogs and the constants that represent the window definition IDs for those types. Note that modeless dialogs are a special case in that a normal document window type is used.

Figure 3. WINDOW TYPES FOR DIALOGS AND ALERTS



The window definition ID is derived by multiplying the resource ID of the WDEF by 16 and adding the variation code to the result, as is shown in the following:

WDEF Resource ID	*Variation Code*	*Window Definition ID (Value)*	*Window Definition ID (Constant)*
`65`	`0`	`65 * 16 + 0 = 1040`	`kWindowPlainDialogProc`

65	1	65 * 16 + 1 = 1041	kWindowShadowDialogProc
65	2	65 * 16 + 2 = 1042	kWindowModalDialogProc
65	3	65 * 16 + 3 = 1043	kWindowMovableModalDialogProc
65	4	65 * 16 + 4 = 1044	kWindowAlertProc
65	5	65 * 16 + 5 = 1045	kWindowMovableAlertProc
65	6	65 * 16 + 6 = 1046	kWindowMovableModalGrowProc
64	0	64 * 16 + 0 = 1024	kWindowDocumentProc (Used for modeless dialogs.)
68	0	68 * 16 + 0 = 1088	kWindowSheetProc

Content of Alerts and Dialogs

Alerts should usually contain only informative text and push button controls. Dialogs may contain informative or instructional text and controls.

Default Push Buttons

Your application should specify a default push button for every alert and dialog. The default push button, visually identified by a default ring drawn around it (Mac OS 8/9) or pulsing blue (Mac OS X), should be the one the user is more likely to click in most circumstances. If the most likely choice is at all destructive (for example, erasing a disk or deleting a file), you should consider defining the Cancel button as the default. (See the caution alert in Fig 1.)

Removing Dialogs

Your application should remove and dispose of a modal, movable modal, and window-modal (sheet) dialogs only when the user clicks one of its push buttons.

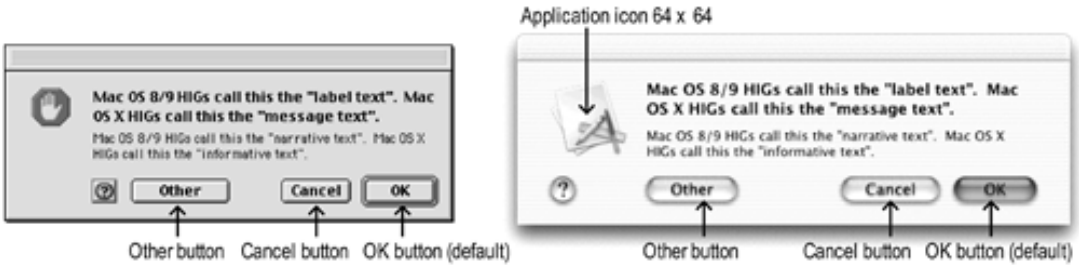
Your application should not remove a modeless dialog unless the user clicks its close box/button or chooses Close from the File menu when the modeless dialog is the active window. (Typically, a modeless dialog is simply hidden, not disposed of, when the user clicks the close box/button or chooses Close from the File menu.)

Creating and Removing Alerts

Alerts may be created from resources using the functions `Alert`, `NoteAlert`, `CautionAlert` and `StopAlert`, which take descriptive information about the alert from alert (`'ALRT'`) and extended alert (`'alrx'`) resources. However, the preferred (and considerably simpler) method is to create alerts programmatically using the functions `StandardAlert` and, on Mac OS X only, `CreateStandardAlert`.

Fig 4 shows an alert created by `StandardAlert` and `CreateStandardAlert`.

Figure 4. ALERTS CREATED WITH StandardAlert AND CreateStandardAlert


Mac OS 8/9 HIGs call this the "label text". Mac OS X HIGs call this the "message text".
Other
Cancel
OK
Other button
Cancel button
OK button (default)
Application icon 64 x 64
Mac OS 8/9 HIGs call this the "label text". Mac OS X HIGs call this the "message text".
Other
Cancel
OK
Other button
Cancel button
OK button (default)

The StandardAlert Function

When an alert is created using the function `StandardAlert`, the alert is automatically sized based on the amount of text passed in to it, and push buttons are automatically sized and located.

```
OSErr  StandardAlert(AlertType inAlertType,ConstStr255Param inError,
                     ConstStr255Param inExplanation,
                     const AlertStdAlertParamRec *inAlertParam,SInt16 *outItemHit);
```

`inAlertType` The level of alert. Relevant constants are:

```
kAlertStopAlert
kAlertNoteAlert
kAlertCautionAlert
kAlertPlainAlert
```

`inError` The label text (Mac OS 8/9) or message text (Mac OS X).

`inExplanation` The narrative text (Mac OS 8/9) or informative text (Mac OS X). `NULL` indicates no narrative/ informative text.

`inAlertParam` A pointer to a **standard alert parameter structure** (see below). `NULL` indicates that none of the features provided by the standard alert structure are required.

`outItemHit` On return, contains the item number of the push button that the user hit.

Standard Alert Parameter Structure

The standard alert parameter structure is as follows:

```
struct AlertStdAlertParamRec
{
```

```
  Boolean        movable;
  Boolean        helpButton;
  ModalFilterUPP filterProc;
  ConstStringPtr defaultText;
  ConstStringPtr cancelText;
  ConstStringPtr otherText;
  SInt16         defaultButton;
  SInt16         cancelButton;
  UInt16         position;
};
typedef struct AlertStdAlertParamRec AlertStdAlertParamRec;
typedef AlertStdAlertParamRec *AlertStdAlertParamPtr;
```

Field Descriptions

`movable` Specifies whether the alert is modal or movable modal.

`helpButton` Specifies whether the alert should include the Help button.

`filterProc` Optionally, a universal procedure pointer to an application-defined event filter (callback) function. If `NULL` is assigned, the Dialog Manager uses the standard event filter (callback) function. (See Event Filter (Callback) Functions For Modal and Movable Modal Alerts and Dialogs later in this chapter.)

`defaultText` Optionally, text for the push button in the OK push button, position. (See Alert Default Text Constants.) The push button is automatically sized and positioned to accommodate the text.

> **Note**
>
> The push button in the OK button position is not necessarily named OK. Human Interface Guidelines require that, wherever possible, buttons be named with a verb that describes the action that they perform. (As an example, see the buttons in Fig 1.)

`cancelText` Optionally, text for the push button in the Cancel push button position. (See Alert Default Text Constants later in this chapter.) The push button is automatically sized and positioned to accommodate the text. Pass `NULL` to specify that a Cancel push button should not be displayed.

`otherText` Optionally, text for the push button in leftmost position. (See Alert Default Text Constants later in this chapter.) The push button is automatically sized and positioned to accommodate the text. Pass `NULL` to specify that the leftmost push button should not be displayed.

`defaultButton` Specifies which push button is to act as the default push button. (See Alert Push Button Constants later in this chapter.)

`cancelButton` Specifies which push button is to act as the Cancel push button. Can be `0`. (See Alert Button Constants later in this chapter.)

`position` The alert position. (See Positioning Specification, below, and note that, when these constants are used to specify alert position in an alert created programmatically using `StandardAlert`, the constant `kWindowDefaultPosition` has the same effect as `kWindowAlertPositionParentWindowScreen`.)

Alert Default Text Constants

To specify the default text for the push buttons in the Right, Middle, and Leftmost push button positions, use these constants in the `defaultText`, `cancelText`, and `otherText` fields of the standard alert structure:

Constant	*Value*	*Button Position*	*Default Text*	*Where Used*
`kAlertDefaultOKText`	`-1`	Right	OK	`defaultText` field
`kAlertDefaultCancelText`	`-1`	Middle	Cancel	`cancelText` field
`kAlertDefaultOtherText`	`-1`	Leftmost	Don't Save	`otherText` field

Alert Push Button Constants

To specify which push buttons act as the default and Cancel push buttons, use these constants in the `defaultButton` and `cancelButton` fields in the standard alert structure:

Constant	*Value*	*Meaning*
`kAlertStdAlertOKButton`	`1`	The OK push button.
`kAlertStdAlertCancelButton`	`2`	The Cancel push button.
`kAlertStdAlertOtherButton`	`3`	A third push button.

Positioning Specification

The main constants for the positioning specification field are as follows:

Constant	*Value*	*Meaning*
`kWindowDefaultPosition`	`0x0000`	Alert position on screen where user is currently working.
`kWindowAlertPositionMainScreen`	`0x300A`	Alert position on main screen. (The main screen in a multi-monitor system is the screen on which the menu bar is located.)
`kWindowAlertPositionParentWindow`	`0xB00A`	Alert position on frontmost window.
`kWindowAlertPositionParentWindowScreen`	`0x700A`	Alert position on screen where user is currently working.
`kWindowCenterMainScreen`	`0x280A`	Center on main screen.
`kWindowCenterParentWindow`	`0xA80A`	Center on frontmost window.
`kWindowCenterParentWindowScreen`	`0x680A`	Center on screen where user is currently working.

The CreateStandardAlert Function

On Mac OS X only, you may also use the function `CreateStandardAlert` to create an alert:

```
OSStatus  CreateStandardAlert(AlertType inAlertType, CFStringRef inError,
                              CFStringRef inExplanation,
                              const AlertStdCFStringAlertParamRec *param,
                              DialogRef *outAlert);
```

The main differences between the `CreateStandardAlert` and `StandardAlert` functions are as follows:

- The `inError` and `inExplanation` fields take a `CFStringRef`.
- A pointer to a standard CFString alert parameter structure is passed in the `param` parameter. This structure is

basically similar to the standard alert parameter structure except that:

- It has no field for a universal procedure pointer to an event filter (callback) function.
- The `defaultText`, `cancelText`, and `otherText` fields are of type `CFStringRef`.
- It has an additional field (`flags`) in which bits can be set to specify options for the behavior of the dialog. Setting the `kStdAlertDoNotDisposeSheet` bit in this field when the dialog is a sheet causes the sheet not to be disposed of after it is hidden.

  A call to the function `GetStandardAlertDefaultParams` initializes a standard CFString alert parameter structure with default values. (The defaults are: not movable, no Help button, no Cancel button, no Other button, alert position on parent window screen.)

- On return, a pointer to a dialog reference is received in the `outAlert` parameter. This must be passed in a call to the function `RunStandardAlert`, which displays the alert and handles user interaction. A universal procedure pointer to an event filter (callback) function may be passed in the `filterProc` parameter of this function. On return, the item number of the push button that the user hit is received in the `outItemHit` parameter of `RunStandardAlert`.

Removal of Alerts

The Dialog Manager automatically removes and disposes of an alert when the user clicks a push button.

Creating Dialogs

Dialogs may be created in one of the following ways:

- You can create dialogs from resources using the function `GetNewDialog`, which takes descriptive information about the dialog from **dialog** (`'DLOG'`) and **extended dialog** (`'dlgx'`) **resources.** The resource ID of the `'DLOG'` and `'dlgx'` resources must be the same, and is passed in the first parameter of this function.

- You can create dialogs programmatically using the function `NewFeaturesDialog`. `NewFeaturesDialog` has a `flags` parameter containing the same flags you would set in an extended dialog resource when creating the dialog from resources.

Regardless of which method is used to create the dialog, a **dialog object** will be created, and a pointer to that object will be returned to the calling function. The dialog object itself includes a window object.

The Dialog Object

Dialog objects are opaque data structures in which the Dialog Manager stores information about individual dialogs. The data type `DialogRef` is defined as a pointer to a dialog object:

```
typedef struct OpaqueDialogPtr *DialogPtr;
typedef DialogPtr DialogRef;
```

Accessor Functions

The following accessor functions are provided to access the information in dialog objects.

Function	*Description*
`GetDialogWindow`	Gets a reference to the dialog's window object.
`GetDialogTextEditHandle`	Gets a handle to the `TERec` structure (which is re-used for all edit text items).
`GetDialogKeyboardFocusItem`	Gets item number of the item with keyboard focus.
`GetDialogDefaultItem`	Gets the item number of the default push button.
`SetDialogDefaultItem`	Tells the Dialog Manager the item number of the default push button.
`GetDialogCancelItem`	Gets the item number of the default Cancel push button.
`SetDialogCancelItem`	Tells the Dialog Manager the item number of the default Cancel button.
`AppendDITL`	Add items to, and remove items from, a dialog.

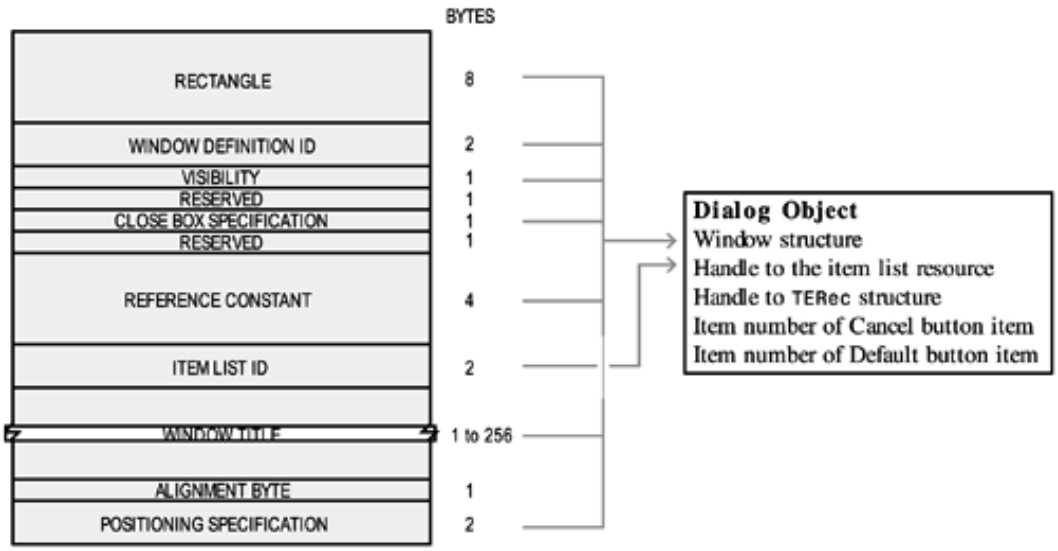
`AppendDialogItemList`
`ShortenDITL`
`InsertDialogItem`
`RemoveDialogItem`	

'DLOG' and 'dlgx' Resources

Structure of a Compiled 'DLOG' Resource

Fig 5 shows the structure of a compiled `'DLOG'` resource and how it "feeds" the dialog object.

Figure 5. STRUCTURE OF A COMPILED DIALOG ('DLOG') RESOURCE



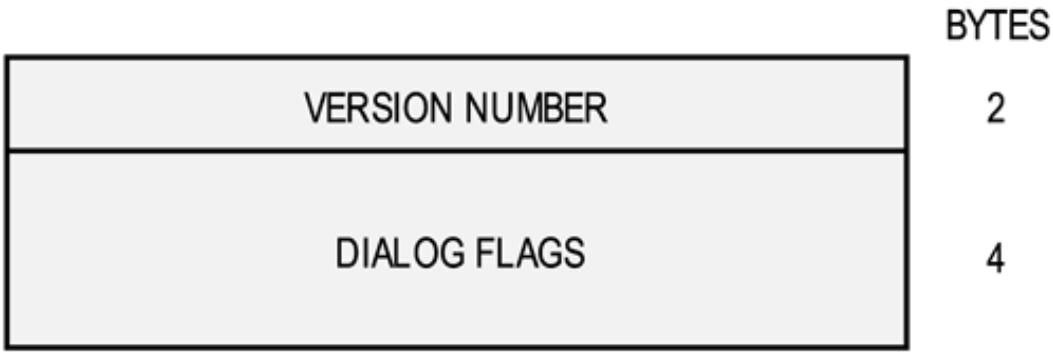
The following describes the main fields of the 'DLOG' resource:

Field	*Description*
RECTANGLE	The dialog's dimensions and, if a positioning specification (see below) is not specified, its location.
WINDOW DEFINITION ID	The window definition ID. (See Window Types For Alerts and Dialogs earlier in this chapter.)
VISIBILITY	If set to 1, the Dialog Manager displays the dialog as soon as `GetNewDialog` is called. If set to 0, the dialog is not displayed until `ShowWindow` is called.
CLOSE BOX SPECIFICATION	Specifies whether to draw a close box/button. Ordinarily, a close box/button is specified only for modeless dialogs.
REFERENCE CONSTANT	Applications can store any value here. For example, an application might store a number that represents the dialog type. `SetWRefCon` and `GetWRefCon` may be used to set and get this value.
ITEM LIST ID	Resource ID of the item list resource.
WINDOW TITLE	The title displayed in the dialog's title bar (modeless and movable modal dialogs only).
POSITIONING SPECIFICATION	Specifies the position of the dialog on the screen. If a positioning constant is not provided, the Dialog Manager places the dialog at the global coordinates specified for the dialog's rectangle (see above). The same positioning constants as apply in the case of an alert apply. (See Positioning Specification earlier in this chapter, but note that, in the case of `'DLOG'` resources, `kWindowDefaultPosition` means that the window will be positioned according to the RECTANGLE field.)

Structure of a Compiled 'dlgx' Resource

Fig 6 shows the structure of a compiled `'dlgx'` resource. This resource allows you to provide additional features for your dialog, including movable modal behavior, background color/pattern, and embedding hierarchies.

Figure 6. STRUCTURE OF A COMPILED DIALOG ('dlgx') RESOURCE



The following describes the main field of the `'dlgx'` resource:

Field	*Description*
DIALOG FLAGS	Constants that specify the dialog's Appearance features.

(See Dialog Feature Flag Constants later in this chapter.)

Dialog Feature Flag Constants

You can set the following bits in the dialog flags field of a `'dlgx'` resource to specify the dialog's features:

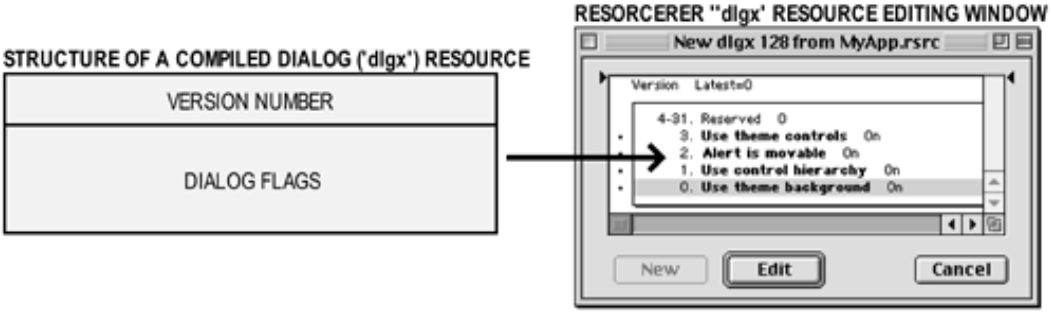
Constant	*Bit*	*Meaning If Set*
`kDialogFlagsUseThemeBackground`	`0`	The Dialog Manager sets the correct dialog background color/pattern.
`kDialogFlagsUseControlHierarchy`	`1`	A root control is created and an embedding hierarchy is established. **Note:** All items in a dialog automatically become controls when embedding hierarchy is established.
`kDialogFlagsHandleMovableModal`	`2`	The dialog will be movable modal (in which case you must use `kWindowMovableModalDialogProc` window definition ID). (The Dialog Manager handles movable modal behavior.)
`kDialogFlagsUseThemeControls`	`3`	All controls created by the Dialog Manager will be compliant with the Platinum appearance.

Creating 'dlgx ' and 'DLOG' Resources Using Resorcerer

Creating 'dlgx' Resources

Fig 7 shows a `'dlgx'` resource being created with Resorcerer.

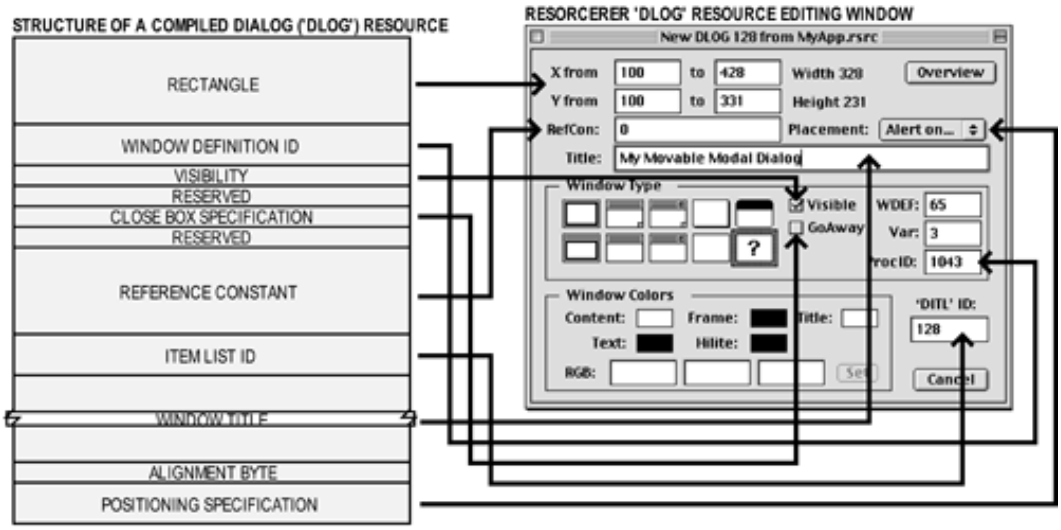
Figure 7. CREATING A 'dlgx' RESOURCE USING RESORCERER



Creating 'DLOG' Resources

Fig 8 shows a `'DLOG'` resource being created with Resorcerer.

Figure 8. CREATING A 'DLOG' RESOURCE USING RESORCERER



The NewFeaturesDialog Function

The function `NewFeaturesDialog` creates a dialog from the information passed in its parameters.

```
DialogRef  NewFeaturesDialog(void *inStorage,const Rect *inBoundsRect,
                             ConstStr255Param inTitle,Boolean inIsVisible,
                             SInt16 inProcID,WindowRef inBehind,Boolean inGoAwayFlag,
                             SInt32 inRefCon,Handle inItemListHandle,UInt32 inFlags);
```

Returns: A pointer to the new dialog, or `NULL` if the dialog is not created

`inStorage` A pointer to the memory for the dialog object. In Carbon, this should always be set to `NULL`, which causes the Dialog Manager to automatically allocate memory for the dialog object.

`inBoundsRect` A rectangle, which specifies the size and position of the dialog in global coordinates.

`inTitle` The title of a modeless or movable modal dialog. You can specify an empty string (not `NULL`) if the dialog is to have no title. (In C, you specify an empty string by two double quotation marks (`""`).)

`inIsVisible` Specifies whether the dialog should be drawn immediately after `NewFeatureDialog` is called. If this parameter is set to `false`, `ShowWindow` must be called to display the dialog.

`inProcID` The window definition ID for the type of dialog. Pass `kWindowModalDialogProc` in this parameter to specify modal dialogs, `kWindowMovableModalDialogProc` to specify movable modal dialogs, and `kWindowDocumentProc` to specify modeless dialogs.

`inBehind` A reference to the window behind which the dialog is to be opened. Pass `(WindowRef) -1` in this parameter to open the dialog in front of all other windows.

`inGoAwayFlag` Passing `true` in this parameter causes a close box/button to be drawn in the title bar. `true` should only be passed when a modeless dialog is being created.

`inRefCon` A reference constant that the Dialog Manager stores in the dialog's window object. Applications can store any

value here. For example, an application might store a number that represents the dialog type. `GetWRefCon` may be used to retrieve this value.

`inItemListHandle` A handle to the item list resource, which you can get by calling `GetResource` to read the item list resource into memory.

`inFlags` The dialog's feature flags. (See Dialog Feature Flag Constants earlier in this chapter.)

Although the `inItemListHandle` parameter specifies an item list (`'DITL'`) resource for the dialog, the corresponding dialog font table (`'dftb'`) resource (see below) is not automatically accessed. You must explicitly set the dialog's control font styles individually.

Items in Dialogs

Preamble-Dialog Manager Primitives

Dialogs contain items, such as push buttons, radio buttons, and checkboxes. Prior to the introduction of Mac OS 8 and the Appearance Manager, an actual control could be an item; however, items such as push buttons and radio buttons were not controls as such but rather **Dialog Manager primitives.**

These primitives may still be specified in item list resources (see following). However, when a root control has been created for the dialog window, thus creating an embedding hierarchy for controls, the Dialog Manager replaces any primitives in the dialog with their control counterparts (except for the primitive called a **user item).**

The situation where all items in a dialog are controls has many advantages. For example, all controls within the dialog can be activated and deactivated by simply activating and deactivating the root control.

The primitives, and their control equivalents, are as follows:

Dialog Manager Primitive	***Control Equivalent***
Button	Push button.
Radio Button	Radio button.
Checkbox	Checkbox.
Edit Text	Edit text control.
Static Text	Static text control.
Icon. (An icon whose black and white resource is stored in an `'ICON'` resource and whose color version is stored in a `'cicn'` resource with the same ID.)	Icon control (no track variant).
Picture. (Picture stored in a `'PICT'` resource.)	Picture control (no track variant).
User Item. (An application-defined item. For example, an application-defined drawing function could be installed in a user item.)	(No control equivalent.)

The 'DITL' Resource

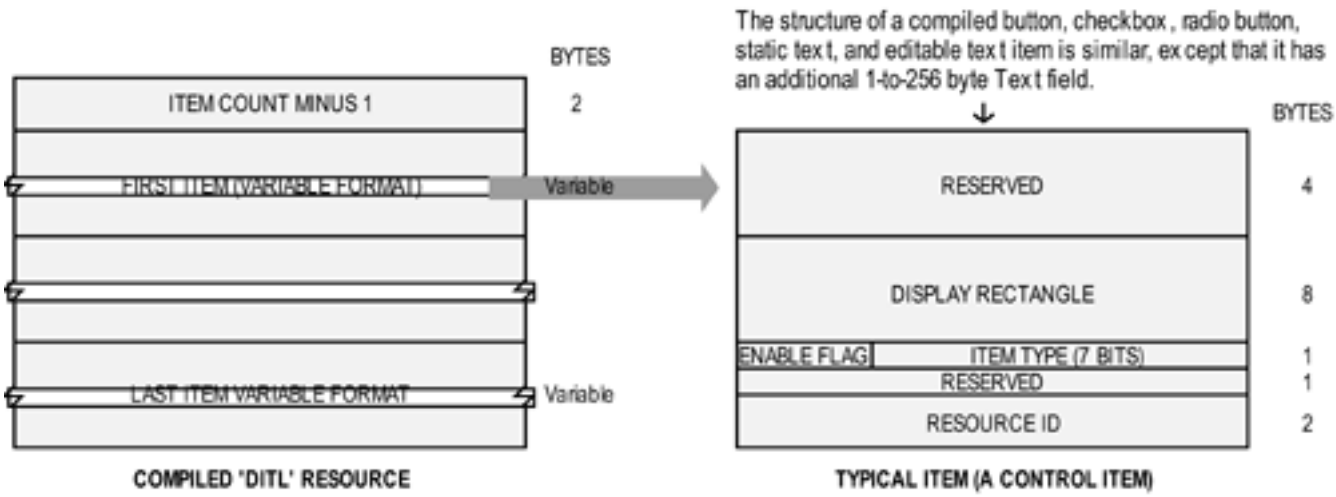
A (`'DITL'`) resource is used to store information about all the items in a dialog. The `'DITL'` resource ID is specified in the associated `'DLOG'` resource, or a handle to the `'DITL'` resource is passed in the `inItemListHandle` parameter of the `NewFeaturesDialog` function. `'DITL'` resources should be marked as purgeable.

Items are usually referred to by their position in the item list, that is, by their item number.

Several independent dialogs may use the same `'DITL'` resource. `AppendDITL`, `AppendDialogItemList`, and `ShortenDITL` may be used to modify or customize copies of shared item list resources for use in individual dialogs.

Fig 9 shows the structure of a compiled `'DITL'` resource and one of its constituent items, in this case a control item.

Figure 9. STRUCTURE OF A COMPILED ITEM LIST ('DITL') RESOURCE AND A TYPICAL ITEM



The following describes the fields of the `'DITL'` resource and the control item:

Field	*Description*
ITEM COUNT MINUS 1	A value equal to one less than the number of items in the resource.
FIRST ITEM … LAST ITEM	(The format of each item depends on its type.)
DISPLAY RECTANGLE	The size and location, in local coordinates, of the item within the dialog. (See Display Rectangles later in this chapter.)

ENABLE FLAG	Specifies whether the item is enabled or disabled. If this bit is set (item is enabled) the Dialog Manager reports all mouse-down events in the item to your application
ITEM TYPE	The item type.
RESOURCE ID	For a control item, the resource ID of the `'CNTL'` resource.

Display Rectangles

The enclosing rectangle you specify in a control's `'CNTL'` resource should be identical to the display rectangle specified in the `'DITL'` resource.

> **Note**
>
> Resorcerer has a Preferences setting which forces conformity between the display rectangle specified in the `'DITL'` resource and the display rectangle specified in the `'CNTL'` resource.

Note that, for items that are controls, the rectangle added to the update region is the rectangle defined in the `'CNTL'` resource, not the display rectangle specified in the `'DITL'` resource. Other important aspects of display rectangles are as follows:

- ***Edit Text Items.*** In edit text items, the display rectangle is the TextEdit **destination rectangle** and **view rectangle** (see Chapter 21). For display rectangles that are large enough to contain more than one line of text, word wrapping occurs within the rectangle. The text is clipped if it overflows the rectangle.

- ***Static Text Item.*** Static text items are drawn within the display rectangle in the same manner as edit text items except that a frame is not drawn around the text.

- ***Icon and Picture Items.*** Icons and pictures larger than the display rectangle are scaled so as to fit the display rectangle.

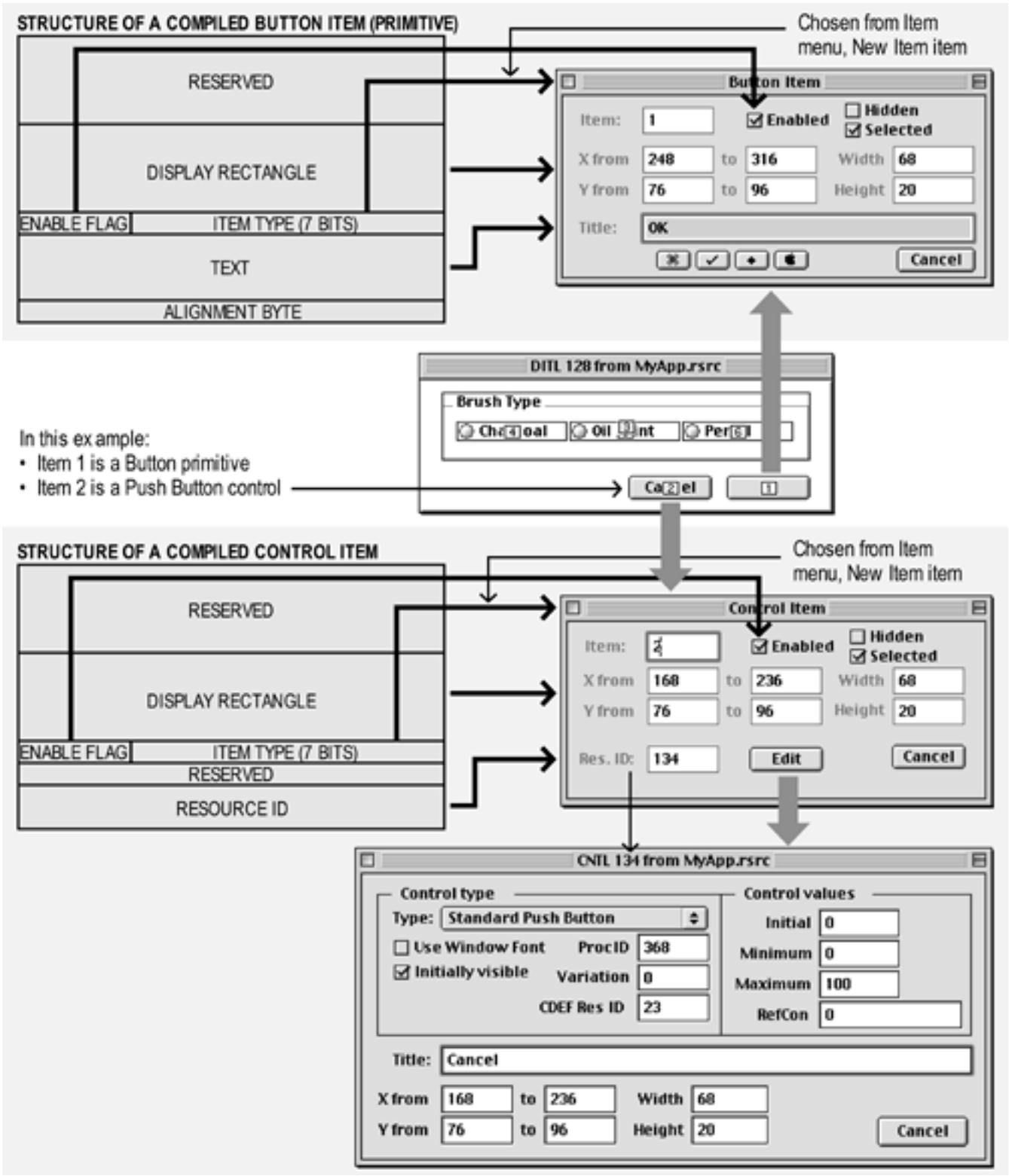
- The Dialog Manager considers a click anywhere in the display rectangle to be a click in that item. In the case of a click in the overlap area of overlapping display rectangles, the Dialog Manager reports the click as occurring in the item that appears first in the item list.

Creating a 'DITL' Resource Using Resorcerer

Fig 10 shows a `'DITL'` resource being created with Resorcerer. Two items are being edited

(Item 1 and Item 2). Item 1 is a Dialog Manager primitive. Item 2 is a control.

Figure 10. CREATING A 'DITL' RESOURCE USING RESORCERER



Layout Guidelines for Dialogs

Layout guidelines for items in dialogs are contained in the Apple publications Mac OS 8

Human Interface Guidelines (Mac OS 8/9) and Aqua Human Interface Guidelines (Mac OS X). These guidelines are not consistent on matters such as the required sizes of certain items and, more particularly, the required spacing between items. For Carbon applications, it is best to observe the Aqua interface guidelines when laying out dialog items.

Default Push Buttons

You should give every dialog a default push button, except for those that contain edit text items that accept Return key presses. If you do not provide an event filter (callback) function (see Event Filter (Callback) Functions For Modal and Movable Modal Alerts and Dialogs later in this chapter) which specifies otherwise, the Dialog Manager treats the first item in the item list resource as the default push button for the purpose of responding to Return and Enter key presses.

Enabling and Disabling Items

You should not necessarily enable all items. For example, you typically disable static text items and edit text items because your application does not need to respond to clicks in those items.

Note that *disabled* is not the same thing as a *deactivated.* The Dialog Manager makes no visual distinction between a disabled and enabled item; it simply does not inform your application when the user clicks a disabled item. On the other hand, when a control is deactivated, the Control Manager dims it to show that it is deactivated.

Keyboard Focus

Edit text and clock items accept input from the keyboard, and list box items respond to certain key presses. The Dialog Manager automatically responds to mouse-down events and Tab key-down events intended to shift the keyboard focus between such items, indicating the current target by drawing a keyboard focus frame around that item. For edit text items, the Dialog Manager also automatically displays the insertion point caret in the current target. For clock items, the Dialog Manager, in addition to drawing the keyboard focus frame, also moves the keyboard target within the clock by highlighting the individual parts.

The Tab key moves the keyboard focus between such items in a sequence determined by their order in the item list. Accordingly, you should ensure that the item numbers of these items in the `'DITL'` resource reflect the sequence in which you require them to be selected by successive Tab key presses.

Manipulating Items

Functions for Manipulating Items

Dialog Manager functions for manipulating items are as follows:

Function	*Description*
`GetDialogItemAsControl`	Returns the control reference for an item in an embedding hierarchy. Should be used instead of `GetDialogItem` (see below) when an embedding hierarchy is established.
`GetDialogItem`	Returns the control reference, item type, and display rectangle of a given item. When an embedding hierarchy is present, you should generally use `GetDialogItemAsControl` instead of `GetDialogItem` to get a reference to the control. When called on a static text item, `GetDialogItem` returns a handle to the text, not a reference to the control, and thus may be used to get a handle to the text of static text items. (When called on a static text item, `GetDialogItemAsControl` returns a reference to the con trol, not a handle to the text.)
`SetDialogItem`	When an embedding hierarchy does not exist, sets the item type, reference, and display rectangle of an item. (When an embedding hierarchy exists, you cannot change the type or reference of an item.)
`HideDialogItem`	Hides the given item.
`ShowDialogItem`	Re-displays a hidden item.
`GetDialogItemText`	Returns the text of an edit or static text item.
`SelectDialogItemText`	Selects the text of an edit text item. When embedding is on, you should pass in the control reference produced by a call to `GetDialogItemAsControl`. When embedding is not on, you should pass in the reference produced by a call to `GetDialogItem`.
`FindDialogItem`	Determines the item number of an item at a particular location in a dialog.
`MoveDialogItem`	Moves a dialog item to a specified location in a window. Ensures that, if the item is a control, the control rectangle and the dialog item rectangle (maintained by the Dialog Manager) are always the same.

`SizeDialogItem`	Resizes a dialog item to a specified size. If the dialog item is a control, the control rectangle and the dialog item rectangle (maintained by the Dialog Manager) are always the same.
`CountDITL`	Counts items in a dialog.
`AppendDITL`	Adds items to the end of the item list.
`AppendDialogItemList`	(See Append Method Constants, below.)
`ShortenDITL`	Removes items from the end of the item list.
`InsertDialogItem`	Inserts an item into the item list.
`RemoveDialogItem`	Removes an item from the item list.
`ParamText`	Substitutes up to four different text strings in static text control items.

Append Method Constants

The `AppendDITL`, `AppendDialogItemList`, and `ShortenDITL` functions are particularly useful in the situation where more than one dialog shares the same `'DITL'` resource and you want to tailor the `'DITL'` for each dialog. When calling `AppendDITL` or `AppendDialogItemList`, you specify a new `'DITL'` resource to append to the relevant dialog's existing `'DITL'` resource. You also specify where the Dialog Manager should display the new items by using one of the following constants in the `AppendDITL` or `AppendDialogItemList` call:

Constant	*Value*	*Description*
`overlay`	`0`	Overlay existing items. Coordinates of the display rectangle are interpreted as local coordinates within the dialog.
`AppendDITLRight`	`1`	Append at right. Display rectangles are interpreted as relative to the upper-right coordinate of the dialog.
`appendDITLBottom`	`2`	Append at bottom. Display rectangles are interpreted as relative to the lower-left coordinate of the dialog.

As an alternative to passing these constants, you can append items relative to an existing item by passing a negative number to `AppendDITL` or `AppendDialogItemList`. The absolute value of this number represents the item relative to which the new items are to be positioned. For example, `-3` would cause the display rectangles of the appended items to be offset from the upper-left corner of item number 3 in the dialog.

To use, at a later time, the unmodified version of a dialog whose contents and (possibly) size have been modified by `AppendDITL` or `AppendDialogItemList`, you should call `ReleaseResource` to release the memory occupied by the appended item list.

Getting and Setting the Text in Edit Text and Static Text Items

Dialog Manager functions for getting text from, and setting the text of, edit text and static text items are as follows:

Function	*Description*
`GetDialogItemText`	Gets a copy of the text in static text and edit text items. Pass in the reference produced by a call to `GetDialogItem`, which gets a handle the text in this instance, not a reference to the control.
`SetDialogItemText`	Sets the text string for static text and edit text items. When embedding is on, you should pass in the control reference produced by a call to `GetDialogItemAsControl`. If embedding is not on, pass in the reference produced by `GetDialogItem`.

The function `ParamText` may also be used to set the text string in a static text item in a dialog. A common example is the inclusion of the window title in static text such as "Save changes to the document … before closing?". In this case, the window's title could be retrieved using `GetWTitle` and inserted by `ParamText` at the appropriate text replacement variable (`^0`, `^1`, `^2` or `^3`) specified in the static text item in the `'DITL'` resource.

Since there are four text replacement variables, `ParamText` can supply up to four text strings for a single dialog.

Setting the Font for Controls in a Dialog-'dftb' Resources

When an embedding hierarchy is established in a dialog, you can specify the initial font settings for all controls in a dialog by creating a dialog font table resource (resource type `'dftb'`) with the same resource ID as the alert or dialog's `'DITL'` resource. When a `'dftb'` resource is read in, the control font styles are set, and the resource is marked purgeable.

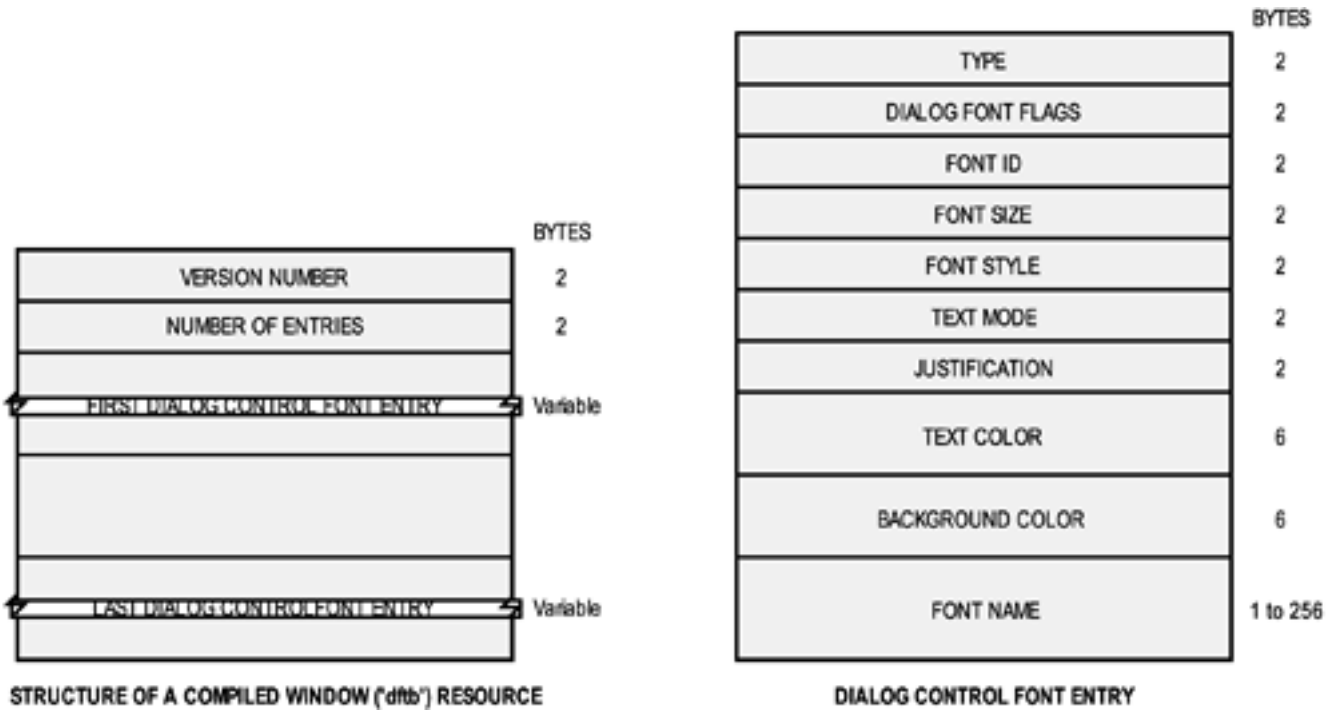
The `'dftb'` resource is the resource-based equivalent of the programmatic method of setting a control's font using the function `SetControlFontStyle` described in Chapter 7.

Structure of a Compiled 'dftb' Resource

Fig 11 shows the structure of a compiled `'dftb'` resource and of a constituent dialog font

table entry.

Figure 11. STRUCTURE OF A COMPILED DIALOG FONT TABLE ('dftb') RESOURCE AND A DIALOG CONTROL FONT ENTRY



The following describes the main fields of the `'dftb'` resource and the dialog control font entry:

Field	***Description***
NUMBER OF ENTRIES	Specifies the number of entries in the resource. Each entry is a dialog control font structure.
FIRST DIALOG CONTROL FONT ENTRY… LASTDIALOG CONTROL	Dialog control font structures. Each comprises type, dialog font flags, font ID, font size, font style, text mode, justification, text color, background color, and font name.
FONT ENTRY TYPE	Specifies whether there is font information for the dialog or alert item in the `'DITL'`. `0` means that there is no font information for the item, that no data follows, and that the entry is to be skipped. `1` means that there is font information for the item, and that the rest of the structure is read.

DIALOG FONT FLAGS	Specifies which of the following fields in the dialog font table should be used. (See Dialog Font Flag Constants later in this chapter.)
FONT ID	The ID of the font family to use. (See Meta Font Constants in the next section for more information about the constants that you can specify.) If this bit is set to `0`, the system default font is used.
FONT SIZE	If the `kDialogFontUseSizeMask` bit in the dialog font flags field is set, the point size of the text. If the `kDialogFontAddSizeMask` bit is set, the size to add to the current point size of the text. If a constant representing the system font, small system font, or small emphasized system font is specified in the Font ID field, this field is ignored.
STYLE	The text style (normal, bold, italic, underlined, outline, shadow, condensed, or extended.)
TEXT MODE	Specifies how characters are drawn. (See Chapter 12 for a discussion of transfer modes.)
JUSTIFICATION	Justification (left, right, centered, or system-justified).
TEXT COLOR	Color to use when drawing the text.
BACKGROUND COLOR	Color to use when drawing the background behind the text. In certain text modes, background color is ignored.
FONT NAME	The font name. This overrides the font ID.

Dialog Font Flag Constants

You can set the following bits in the dialog font flags field of a dialog control font entry to specify the fields in the entry that should be used

Constant	*Value*	*Meaning*
`kDialogFontNoFontStyle`	`0x0000`	No font style information is applied.
`kDialogFontUseFontMask`	`0x0001`	The specified font ID is applied.
`kDialogFontUseFaceMask`	`0x0002`	The specified font style is applied.
`kDialogFontUseSizeMask`	`0x0004`	The specified font size is applied.

`kDialogFontUseForeColorMask`	`0x0008`	The specified text color is applied. This flag only applies to static text controls.
`kDialogFontUseBackColorMask`	`0x0010`	The specified background color is applied. This flag only applies to static text controls.
`kDialogFontUseModeMask`	`0x0020`	The specified text mode is applied.
`kDialogFontUseJustMask`	`0x0040`	The specified text justification is applied.
`kDialogFontUseAllMask`	`0x00FF`	All flags in this mask will be set except `kDialogFontAddFontSizeMask` and `kDialogFontUseFontNameMask`.
`kDialogFontAddFontSizeMask`	`0x0100`	The specified font size will be added to the existing font size specified in the Font Size field of the dialog font table resource.
`kDialogFontUseFontNameMask`	`0x0200`	The string in the Font Name field will be used for the font name instead of the specified font ID.

Meta Font Constants

You can use the following meta font constants in the font ID field of a dialog control font entry to specify the style, size, and font family of a control's font. You should use these meta font constants whenever possible because, on Mac OS 8/9, the system font can be changed by the user. If none of these constants are specified, the control uses the system font unless a control with a variant that uses the window font has been specified.

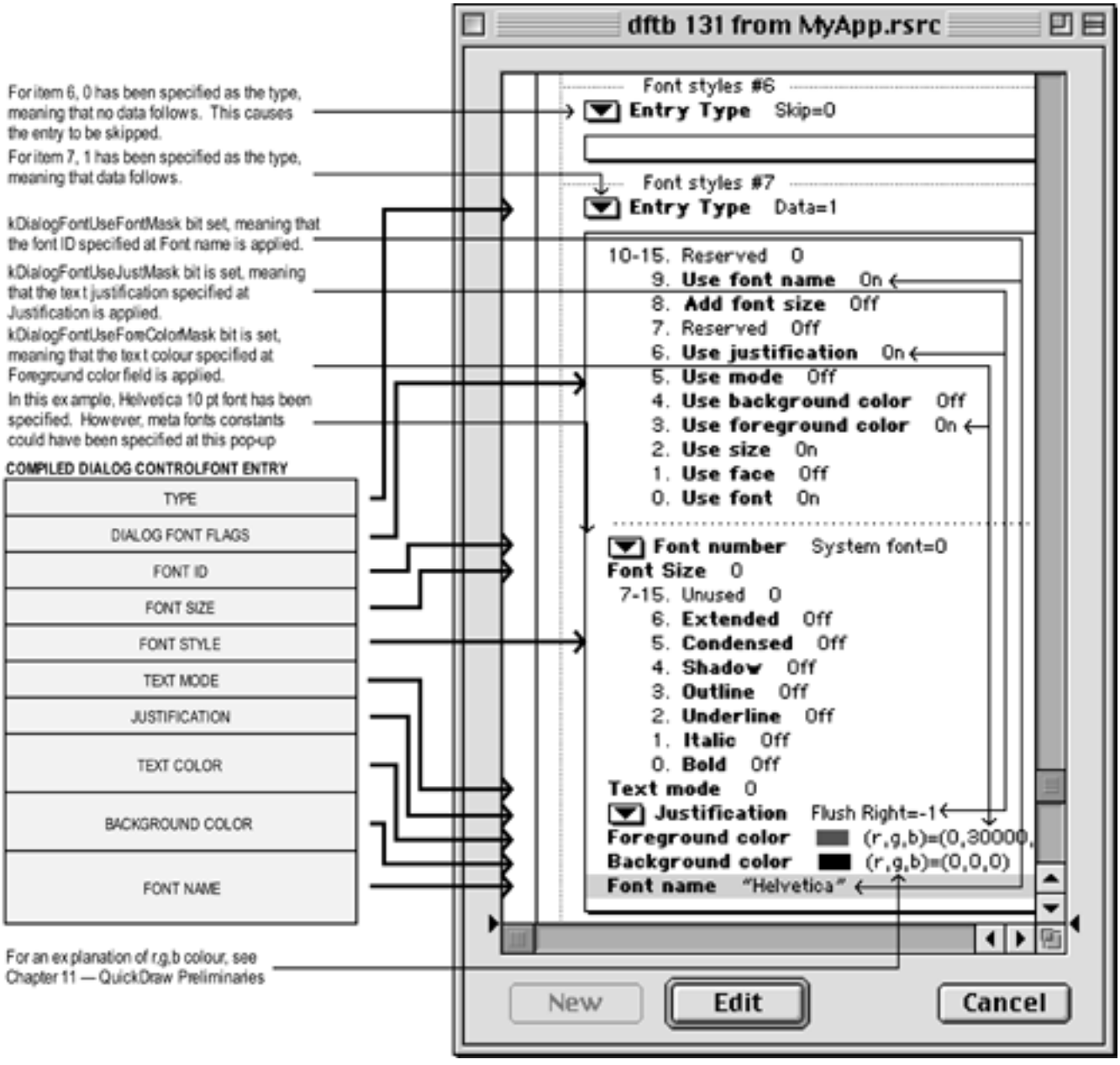
Constant	*Value*	*Meaning In Roman Script System*
`kControlFontBigSystemFont`	`-1`	Use the system font.
`kControlFontSmallSystemFont`	`-2`	Use the small system font.
`kControlFontSmallBoldSystemFont`	`-3`	Use the small bold system font.
`kControlFontSmallBoldSystemFont`	`-4`	Use the small emphasized system font.

Another advantage of using these meta font constants is that you can be sure of getting the correct font on a Macintosh using a different script system, such as kanji.

Creating a 'dftb' Resource Using Resorcerer

Fig 12 shows a dialog control font entry in a `'dftb'` resource being edited with Resorcerer.

Figure 12. CREATING A 'dftb' RESOURCE USING RESORCERER


dftb 131 from MyApp.rsrc
For item 6, 0 has been specified as the type, meaning that no data follows. This causes the entry to be skipped.
For item 7, 1 has been specified as the type, meaning that data follows.
kDialogFontUseFontMask bit set, meaning that the font ID specified at Font name is applied.
kDialogFontUseJustMask bit is set, meaning that the text justification specified at Justification is applied.
kDialogFontUseForeColorMask bit is set, meaning that the text colour specified at Foreground color field is applied.
In this example, Helvetica 10 pt font has been specified. However, meta fonts constants could have been specified at this pop-up
COMPILED DIALOG CONTROLFONT ENTRY
TYPE
DIALOG FONT FLAGS
FONT ID
FONT SIZE
FONT STYLE
TEXT MODE
JUSTIFICATION
TEXT COLOR
BACKGROUND COLOR
FONT NAME
For an explanation of r,g,b colour, see Chapter 11 — QuickDraw Preliminaries
Font styles #6
Entry Type Skip=0
Font styles #7
Entry Type Data=1
10-15. Reserved 0
9. Use font name On
8. Add font size Off
7. Reserved Off
6. Use justification On
5. Use mode Off
4. Use background color Off
3. Use foreground color On
2. Use size On
1. Use face Off
0. Use font On
Font number System font=0
Font Size 0
7-15. Unused 0
6. Extended Off
5. Condensed Off
4. Shadow Off
3. Outline Off
2. Underline Off
1. Italic Off
0. Bold Off
Text mode 0
Justification Flush Right=-1
Foreground color (r,g,b)=(0,30000,
Background color (r,g,b)=(0,0,0)
Font name "Helvetica"
New
Edit
Cancel

Displaying Alerts and Dialogs

As previously stated

- `StandardAlert`, `CreateStandardAlert` and `RunStandardAlert` are used to create and display alerts.
- `GetNewDialog` is used to create dialogs using descriptive information supplied by `'DLOG'` and `'dlgx'` resources, and `NewFeaturesDialog` is used to create dialogs programmatically. Both creation methods allow you to specify whether the dialog is to be initially visible, and both allow you to specify whether the dialog is to be brought to the front of all other windows when it is opened.

To display a dialog that is specified to be invisible on creation, you must call `ShowWindow` following the `GetNewDialog` or `NewFeaturesDialog` call to display the dialog. In addition, you should invariably pass `(WindowRef) -1` in the `behind` and `inBehind` and parameters of, respectively, `GetNewDialog` and `NewFeaturesDialog` call so as to display a dialog as the active (frontmost) window.

Window Deactivation and Menu Adjustment

When an alert or dialog is displayed

- The frontmost window (assuming one exists) must be deactivated.
- The application's menus must be adjusted to reflect the differing levels of permitted menu access that apply in the presence of the various types of alert and dialog. (As will be seen, the system software automatically performs some of this menu adjustment for you.)

  Note

  Prior to the introduction of Mac OS 8 and the Appearance Manager, window deactivation when a movable modal dialog was displayed was handled in the same way as applies in the case of a modeless dialog, that is, within the application's main event loop. However, with the introduction of the Appearance Manager, when the `kDialogFlagsHandleMovableModal` bit is set in the `'dlgx'` resource, or in the `inFlags` parameter of `NewFeaturesDialog`, `ModalDialog` is used to handle all user interaction within the dialog. (Previously, this user interaction was handled within the main event loop.) This has implications for the way your application deactivates the front window when a movable modal dialog is displayed.

Prior to the introduction of Mac OS 8 and the Appearance Manager, menu adjustment when a movable modal dialog was displayed was performed by the application. However, when a movable modal dialog is created by setting the `kDialogFlagsHandleMovableModal` bit in the `'dlgx'` resource, or in the `inFlags` parameter of `NewFeaturesDialog`, menu adjustment is performed by the Dialog Manager and Menu Manager.

All that follows assumes that the `kDialogFlagsHandleMovableModal` bit is set in the `'dlgx'` resource, or in the `inFlags` parameter of `NewFeaturesDialog`, and that, as a consequence:

- Your application calls `ModalDialog` to handle all user interaction within movable modal dialogs (as is the case with modal dialogs).
- Menu adjustment will be performed automatically by the Dialog Manager and Menu Manager when a movable modal dialog is displayed (as is the case with modal dialogs).

Window Deactivation-Modeless Dialogs

You do not have to deactivate the front window *explicitly* when displaying a modeless dialog. The Event Manager continues sending your application activate events for your windows as needed, which you typically handle in your main event loop.

Window Deactivation-Modal and Movable Modal Alerts and Dialogs

When a modal or movable modal alert or dialog is created and displayed, your application (in the case of dialogs) or `StandardAlert` and `RunStandardAlert` (in the case of alerts) calls `ModalDialog` to handle all user interaction within the alert or dialog until the alert or dialog is dismissed. Events, which are ordinarily handled within your application's main event loop, will then be trapped and handled by `ModalDialog`. This means that your window activation/deactivation function will not now be called as it normally would following the opening of a new window. Accordingly, if one of your application's windows is active, you must *explicitly* deactivate it before displaying a modal or movable modal alert or dialog.

Menu Adjustment-Modeless Dialogs

When your application displays a modeless dialog, it is responsible for all menu disabling and enabling. Your application should thus perform the following tasks:

- Disable those menus whose items are not relevant to the modeless dialog.

- For modeless dialogs that contain edit text controls, enable the Edit menu and support the Cut, Copy, Paste, and Clear items using the Dialog Manager functions `DialogCut`, `DialogCopy`, `DialogPaste` and `DialogDelete`.

Your application is also responsible for all menu enabling when a modeless dialog is dismissed.

Menu Adjustment-Modal Alerts and Dialogs

When your application displays a modal alert or dialog, the Dialog Manager and Menu Manager interact to provide varying degrees of access to the menus in your menu bar, as follows:

- On Mac OS 8/9, the Mac OS 8/9 Application menu, and all items in the Help menu except the Show Balloons/Hide Balloons items are disabled. On Mac OS X, all but the Apple and Application menus are disabled.
- Your application's menus are disabled.
- If the modal dialog contains a visible and active edit text item, the Edit menu and its Cut, Copy, and Paste items are enabled.

When the user dismisses the modal alert or dialog, the Menu Manager restores all menus to their previous state.

Menu Adjustment-Movable Modal Alerts and Dialogs

When your application displays a movable modal alert or dialog, the Dialog Manager and Menu Manager interact to provide the same access to the menus in your menu bar as applies in the case of modal alerts and dialogs except that, in this case, on Mac OS 8/9, the Help and Mac OS 8/9 Application menus are left enabled.

When the user dismisses the movable modal alert or dialog, the Menu Manager restores all menus to their previous state.

Displaying Multiple Alerts and Dialogs

The user should never see more than one modal dialog and one modal alert on the screen simultaneously. However, you can present multiple simultaneous modeless dialogs just as you can present multiple document windows.

Resizing a Dialog

You can use the function `AutoSizeDialog` to automatically resize static text items and

their dialogs to accommodate changed static text. For each static text item found, `AutoSizeDialog` adjusts the static text control and the bottom of the dialog window. Any items below a static text control are moved down.

Handling Events in Alerts and Dialogs

Overview

Modal and Movable Modal Alerts and Dialogs

When `StandardAlert`, `CreateStandardAlert`, and `RunStandardAlert` are used to create and display alerts, the Dialog Manager handles all of the events generated by the user until the user clicks a push button. When the user clicks a push button, these functions highlight the push button briefly, close the alert and report the user's selection to the application.

As previously stated, `ModalDialog` handles all user interaction within modal and movable modal dialogs. When the user selects an enabled item, `ModalDialog` reports that the user selected the item and then exits. Your application is then responsible for performing the appropriate action in relation to that item. Your application typically calls `ModalDialog` repeatedly until the user dismisses the dialog.

The `filterProc` field of the standard alert structure associated with the `StandardAlert` function, the `filterProc` parameter of the `RunStandardAlert` function, and the `modalFilter` parameter of the `ModalDialog` function take a universal procedure pointer to a callback function known as an **event filter function.** The Dialog Manager provides a **standard event filter function,** which is used if `NULL` is passed in the `filterProc` or `modalFilter` parameters or assigned to the `filterProc` field; however, you should supply an application-defined event filter (callback) function for modal and movable modal alerts and dialogs so as to avoid a basic limitation of the standard event filter (callback) function. (See Event Filter (Callback) Functions For Modal and Movable Modal Alerts and Dialogs later in this chapter.)

Modeless Dialogs

For modeless dialogs, you can use the function `IsDialogEvent` to determine whether the event occurred while a modeless dialog was the frontmost window and then, optionally, use the function `DialogSelect` to handle the event if it belongs to a modeless dialog. `DialogSelect` is similar to `ModalDialog` except that it returns control after every event, not just events relating to an enabled item. Also, `DialogSelect` does not pass events to an event filter (callback) function.

Responding to Events in Controls

Controls and Control Values

For clicks in those types of controls for which you need to determine or change the control's value, your application should use the Control Manager functions `GetControlValue` and `SetControlValue` to get and set the value. When the user clicks on the OK push button, your application should perform whatever action is necessary to reflect to the values returned by the controls.

Controls That Accept Keyboard Input

Edit text controls and clock controls, which both accept keyboard input, are typically disabled because you generally do not need to be informed every time the user clicks on one of them or types a character. Instead, you simply need to retrieve the text in the edit text control, or the clock's date/time value, when the user clicks the OK push button.

When you use `ModalDialog` (key-down events in edit text controls and clock controls in modal or movable modal dialogs) or `DialogSelect` (key-down events in edit text controls and clock controls in modeless dialogs), keystrokes and mouse actions within those controls are handled automatically. In the case of an edit text control, this means that

- A blinking vertical bar, called the **insertion point caret,** appears when the user clicks the item.
- When the user drags over text or double-clicks a word, that text is highlighted and replaced by whatever the user types.
- Highlighting of text is extended or shortened when the user holds down the Shift key while clicking or dragging.
- Highlighted text, or the character preceding the insertion point caret, is deleted when the user presses the backspace key.
- Highlighted text, or the character following the insertion point caret, is deleted when the user presses the delete key.
- When the user presses the Tab key, the cursor and keyboard focus frame automatically advance to the next edit text

control, clock control, or list box (if any) in the item list, wrapping around to the first one if there are no more items.

Caret Blinking in Edit Text Controls

`ModalDialog` will cause the insertion point caret to blink in edit text controls in modal and movable modal dialogs. On Mac OS 8/9, for edit text controls in a modeless dialog, you should call `IdleControls` in your main event loop's idle processing function. (This is not necessary on Mac OS X because controls on Mac OS X have their own built-in timers.) `IdleControls` calls the edit text control with an idle event so that the control can call `TEIdle` to make the insertion point caret blink. You should ensure that, when caret blinking is required, the `sleep` parameter in the `WaitNextEvent` call is set to a value no greater that that returned by `GetCaretTime`.

Responding to Events in Modal and Movable Modal Alerts

`StandardAlert` and `RunStandardAlert` handle events automatically, calling `ModalDialog` internally.

If the event is a mouse-down anywhere outside the content region of a modal alert, `ModalDialog` emits the system alert sound and gets the next event.

If the event is a mouse-down outside the content region of a movable modal alert and within a window belonging to the application, `ModalDialog` emits the system alert sound and gets the next event. If the mouse-down is not within the content region or a window belonging to the application, `ModalDialog` performs alert dragging (if the mouse-down is within the title bar) or sends the application to the background (if the mouse-down is not within the title bar).

`ModalDialog` is continually called until the user clicks an enabled control, at which time `StandardAlert` and `RunStandardAlert` remove the alert from the screen and return the item number of the selected control. Your application then should respond appropriately.

The standard event filter (callback) function allows users to use the Return or Enter key to achieve the same effect as a click on the default push button. When you write your own event filter (callback) function, you should ensure that that function retains this behavior. `ModalDialog` passes events inside the alert to your event filter (callback) function *before* handling the event. Your event filter (callback) function thus provides a means to

- Handle events that `ModalDialog` does not handle.
- Override events `ModalDialog` would otherwise handle.

If your event filter (callback) function does not handle an event inside an alert in its own way, `ModalDialog` handles the event as follows:

- For activate or update events, `ModalDialog` activates or updates the alert window.
- For mouse-down events in a trackable control, `TrackControl` is called to track the mouse. If the user releases the mouse button while the cursor is still in the control, the alert is removed and the control's item number is returned.
- For a mouse-down event in a disabled item, or in no item, or if any other event occurs, nothing happens.

Responding to Events in Modal and Movable Modal Dialogs

Your application should call `ModalDialog` immediately after displaying a modal or movable modal dialog. `ModalDialog` repeatedly handles events inside the dialog until an event involving an enabled item occurs, at which time `ModalDialog` exits, returning the item number. Your application should then respond appropriately. `ModalDialog` should be continually called until the user clicks on the OK, Cancel, or Don't Save push button, at which time your application should close the dialog.

If the event is a mouse-down anywhere outside the content region of a modal dialog, `ModalDialog` emits the system alert sound and gets the next event.

If the event is a mouse-down outside the content region of a movable modal dialog and within a window belonging to the application, `ModalDialog` emits the system alert sound and gets the next event. If the mouse down is not within the content region or a window belonging to the application, `ModalDialog` performs dialog dragging (if the mouse-down is within the title bar) or sends the application to the background (if the mouse-down is not within the title bar).

If your event filter (callback) function does not handle the event, `ModalDialog` handles the event as follows:

- For activate or update events, `ModalDialog` activates or updates the dialog window.
- If the event is a mouse-down while the cursor is in a control that accepts keyboard input (that is, an edit text control or a clock control), `ModalDialog` responds to the mouse activity by either displaying an insertion point or by selecting text in an edit text control or by highlighting the appropriate part of the clock control. Where there is more than one control that accepts keyboard input, `ModalDialog` moves the keyboard focus to that control. If a key-down event occurs and there is an edit text control in the dialog, `ModalDialog` uses TextEdit to handle text entry and editing automatically. For an enabled edit text control, `ModalDialog` returns its item number after it receives either the mouse-down or key-down event. (Normally, edit text controls should be disabled.)
- For mouse-down events in a trackable control, `TrackControl` is called to track the mouse. If the user releases the mouse button while the cursor is still in the control, the control's item number is returned.
- If the event is a Tab key key-down event and there is more than one control that accepts keyboard input, `ModalDialog` moves the keyboard focus to the next such item in the item list.
- For a mouse-down event in a disabled item, or in no item, or if any other event occurs, nothing happens.

Specifying the Events To Be Received by ModalDialog

The function `SetModalDialogEventMask` may be used to specify the events to be received by the `ModalDialog` function for a given modal or movable modal dialog. This allows your application to specify additional events that are not by default received by `ModalDialog`, such as operating system events. If you use this function to change the `ModalDialog` function's event mask, you must pass `ModalDialog` a universal procedure pointer to your own event filter (callback) function to handle the added events.

You can ascertain the events to be received by `ModalDialog` by calling `GetModalDialogEventMask`.

Simulating Item Selection

You can cause the Dialog Manager to simulate item selection in a modal or movable modal dialog using the function `SetDialogTimeout`. You can use this function in circumstances where you wish to start a countdown for a specified duration for a specified dialog. When the specified time elapses, the Dialog Manager simulates a click on the specified button. The Dialog Manager will not simulate item selection until `ModalDialog` processes an event.

You can ascertain the original countdown duration, the time remaining, and the item selection to be simulated by calling `GetDialogTimeout`.

Event Filter (Callback) Functions for Modal and Movable Modal Alerts and Dialogs

The standard event filter (callback) function dates from the early days of the Macintosh, when a single application controlled the computer. With the introduction of multitasking, however, the standard event filter proved to be somewhat inadequate, its main deficiency being that it does not cater for the updating of either the parent application's windows or those belonging to background applications. (This deficiency is only relevant on Mac OS 8/9.) Your application should therefore provide an application-defined event filter (callback) function that compensates for this inadequacy and handles other events you wish the function to handle.

The standard event filter (callback) function performs the following checks and actions:

- Checks whether the user has pressed the Return or Enter key and, if so, highlights the default push button for eight ticks (Mac OS 8/9 only) and returns the item number of that push button. (Unless informed otherwise, the Dialog Manager assumes that the first item in the item list is the default push button.)
- For dialogs only, and only if the application has previously called certain Dialog Manager functions (see below)
    - Checks whether the user has pressed the Escape key or Command-period and, if so, highlights the Cancel push button for eight ticks (Mac OS 8/9 only) and returns the item number of that button.
    - Check whether the cursor is over an edit text item and, if so, changes the cursor shape to the I-Beam cursor.

As a minimum, your application-defined event filter (callback) function should ensure that these checks and actions are performed and should also

- For Mac OS 8/9 only, handle update events not belonging to the alert or dialog so as to allow the application to update its own windows, and return `false`. (Note that, by responding to update events in the application's own windows in this way, you also allow `ModalDialog` to perform a minor switch when necessary so that background applications can update their windows as well.)
- Return `false` for all events that your event filter (callback) function does not handle.

Defining an Event Filter (Callback) Function

Part of the recommended approach to defining a basic event filter (callback) function is to continue to use the standard event filter (callback) function to perform its checks and actions as described earlier. This requires certain preliminary action which, for dialogs, requires calls similar to the following examples after the dialog is created and before the call to `ModalDialog`:

```
// Tell the Dialog Manager which is the default push button item, alias the Return and
// Enter keys to that item, and draw the default ring around that item (Mac OS 8/9) or
// make it pulsing blue (Mac OS X).

SetDialogDefaultItem(myDialogRef,iOK);

// Tell the Dialog Manager which is the Cancel push button item, and alias the escape
// key and Command-period key presses to that item.

SetDialogCancelItem(myDialogRef,iCancel);

// Tell the Dialog Manager to track the cursor and change it to the I-Beam cursor shape
// whenever it is over an edit text item.

SetDialogTracksCursor(myDialogRef,true);
```

Note that it is essential that the default and Cancel push buttons, and edit text items, be specified as primitives, not as actual controls, in the `'DLOG'` resource.

With those preparations made, you would define your basic event filter (callback) function as in the following example:

```
Boolean  myEventFilterFunction(DialogRef dialogRef,EventRecord *eventStrucPtr,
                               SInt16 *itemHit)
{
  Boolean handledEvent;
  GrafPtr oldPort;

  handledEvent = false;

  if((eventStrucPtr->what == updateEvt) &&
     ((WindowRef) eventStrucPtr->message != dialogRef))
  {
    // If the event is an update event, and if it is not for the dialog or alert, call
    // your application's window updating function, and return false.
    if(!gRunningOnX)
      doUpdate(eventStrucPtr);
  }
  else
  {
    // If the event was not an update, first save the current graphics port and set the
    // alert or dialog's graphics port as the current graphics port. This is
    // necessary when you have called SetDialogTrackCursor to cause the Dialog Manager
    // to track cursor position.

    GetPort(&oldPort);
    SetPortDialogPort(dialogRef);

    // Pass the event to the standard event filter function for handling. If the
    // function handles the event, it will return true and, in the itemHit parameter,
```

```
    // the number of the item that it handled. ModalDialog, StandardAlert, and
    // RunStandardAlert then return this item number in their own itemHit parameter.

    handledEvent = StdFilterProc(dialogRef,eventStrucPtr,itemHit);

    // Make the saved graphics port the current graphics port again.

    SetPort(oldPort);
  }

  // Return true or false, as appropriate.

  return(handledEvent);
}
```

`StandardAlert`, `RunStandardAlert`, and `ModalDialog` pass events to your event filter (callback) function *before* handling each event, and will handle the event if your event filter (callback) function returns `false`.

> **Note**
>
> A major difference between modal alerts and dialogs and movable modal alerts and dialogs is that, in the case of the latter, *all* events are passed to your event filter function for handling. This allows you to, for example, handle suspend and resume events when your application is either moved to the background or brought to the front, as well as other events you might want to handle.

You can also use your event filter (callback) function to handle events that `ModalDialog` does not handle, such as keyboard equivalents and mouse-down events.

Responding to Events in Modeless Dialogs

As previously stated, you can use the function `IsDialogEvent` to determine whether an event occurred in a modeless dialog or a document window and then call `DialogSelect` to handle the event if it occurred in a modeless dialog. `DialogSelect` handles the event as follows:

- For activate or update events, `DialogSelect` activates or updates the modeless dialog and returns `false`.
- If the event is a key-down or auto-key event, and there is an edit text item in the modeless dialog, `DialogSelect` uses TextEdit to handle text entry and editing and returns `true` and the item number. If there is no edit-text item, `DialogSelect` returns `false`.
- For mouse-downs in an edit text item, `DialogSelect` displays the insertion point caret or selects text as appropriate. `DialogSelect` returns `false` if the edit text item is disabled, and `true` and the item number if it is enabled. (Normally, edit text items should be disabled.)
- For mouse-downs in an enabled trackable control, `DialogSelect` calls `TrackControl` and, if the user releases the mouse button while the cursor is still within the control, returns `true` and the item number.
- For mouse-downs on a disabled item, or in no item, or if any other event occurs, `DialogSelect` does nothing.

In the case of a key-down or auto-key event in an edit text item, you will ordinarily need to filter out Return and Enter key presses and certain Command-key equivalents so that they are not passed to `DialogSelect`. In the case of Return and Enter key presses, you should also highlight the associated push button for eight ticks (for Mac OS 8/9) before calling the function which responds to hits on the OK button. In the case of Command-key presses, you should only allow Command-X, Command-C, and Command-V to be passed to `DialogSelect` (so that `DialogSelect` can support cut, copy, and paste actions within the edit text control) and pass any other Command-key equivalents to your application's menu handling function.

Closing and Disposing of Dialogs

`CloseDialog` closes the dialog's window and removes it from the screen, and frees up the memory occupied by most types of items in the item list and related data structures. It does not release the memory occupied by the dialog object or item list.

`DisposeDialog` closes the dialog's window and deletes it from the window list, and releases the memory occupied by the dialog object, item list, and most types of items. (Handles leading to icons and pictures are not released.)

For modeless dialogs, you might find it more efficient to hide the dialog with HideWindow rather than dispose of the dialog. In that way, the dialog will remain available, and in the same location and with the same settings as when it was last used.

Creating Displaying and Handling Window-Modal (Sheet) Alerts and Dialogs

Window-Modal (Sheet) Alerts

Window-modal (sheet) alerts are created using the function `CreateStandardSheet`:

```
OSStatus  CreateStandardSheet(AlertType alertType,CFStringRef error,
                              CFStringRef explanation,
                              const AlertStdCFStringAlertParamRec *param,
                              EventTargetRef notifyTarget, DialogRef *outSheet);
```

`alertType` The level of the alert. Relevant constants are

```
kAlertStopAlert
kAlertNoteAlert
kAlertCautionAlert
kAlertPlainAlert
```

`error` The message text.

`explanation` The informative text.

`param` A pointer to a **standard CFString alert parameter structure** (see above). `NULL` indicates that none of the features provided by the standard alert structure are required.

`notifyTarget` The event target to be notified when the sheet is closed.

`outSheet` On return, the sheet's dialog reference.

The sheet will be invisible when created. A call to `ShowSheetWindow` displays the sheet.

If the sheet has more than one button, your application will need to determine which button was hit by the user. This requires the use of the Carbon event model and the installation of an event handler on the event target (see Chapter 17). The event target is ordinarily the owner window, in which case you pass a reference to that window in a call to `GetWindowEventTarget` and pass the returned event target reference in the `notifyTarget` parameter of `CreateStandardSheet`. The Carbon event handler you install should respond to the `kEventProcessCommand` Carbon event type and should test for the command IDs `kHICommandOK`, `kHICommandCancel`, and `kHICommandOther` in order to determine which button was hit.

If the sheet has only one button (an OK button), you can simply pass the returned event target reference in the `notifyTarget` parameter (so that the `CreateStandardSheet` call will not fail) and not install a handler. The sheet will be dismissed when the button is hit.

Window-Modal (Sheet) Dialogs

Window-modal (sheet) dialogs, like other dialogs, may be created using `GetNewDialog` or `NewFeaturesDialog`. The window definition ID should be `kWindowSheetProc` (`1088`) and the dialog should be created invisible.

A call to `ShowSheetWindow` displays the sheet. Your application should ensure that only one sheet is displayed in a

window at one time.

Events in window-modal (sheet) dialogs may be handled in the same way as for modeless dialogs.

Balloon Help for Dialogs-Mac OS 8/9

Two basic options are available for adding help balloons to dialogs for Mac OS 8/9:

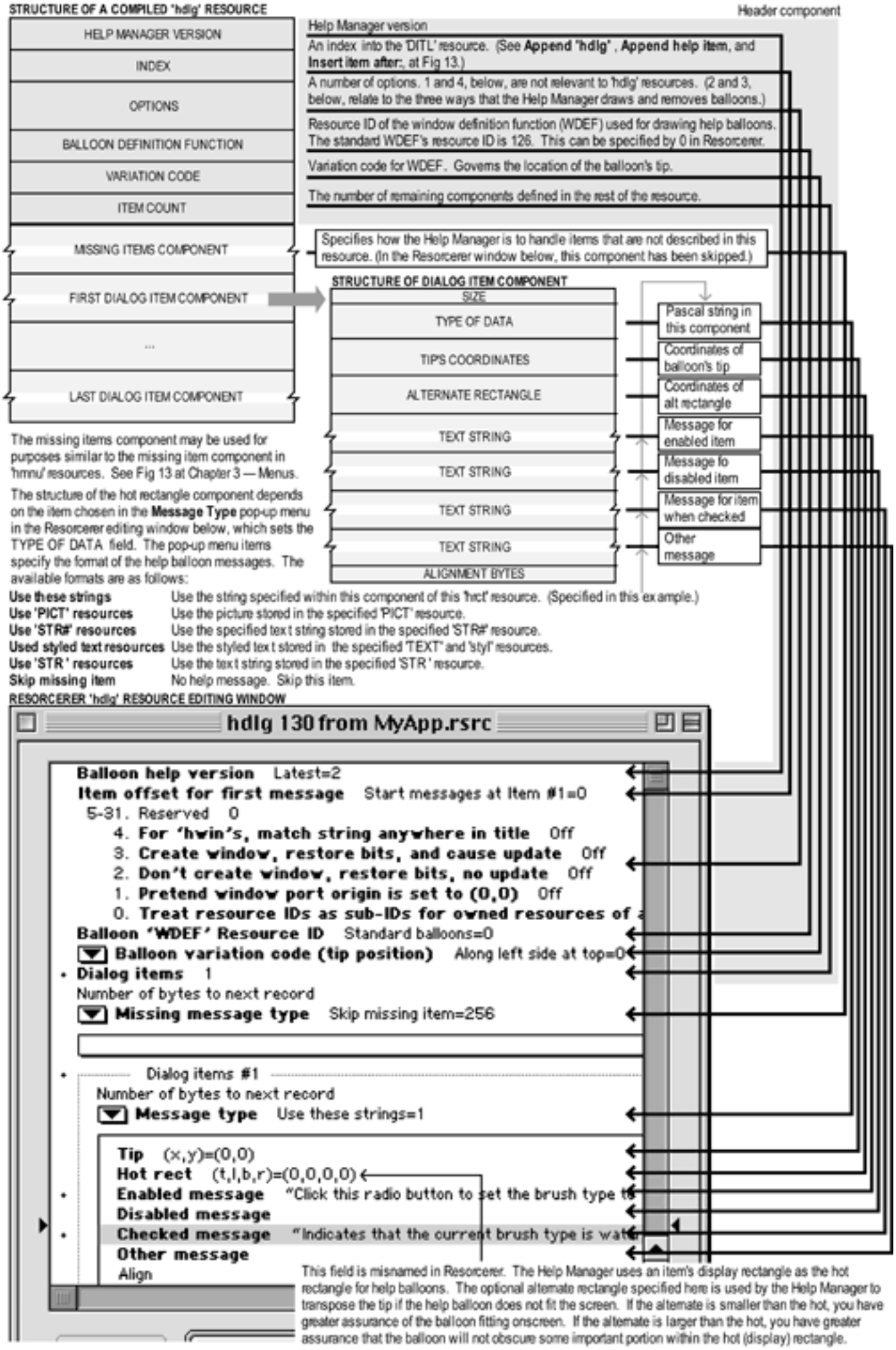
- Adding a **balloon help item** to the item list (`'DITL'`) resource, which will associate either a **rectangle help** (`'hrct'`) resource or a **dialog help** (`'hdlg'`) resource with that `'DITL'` resource. Each hot rectangle component in the `'hrct'` resource, and each dialog item component in the `'hdlg'` resource, corresponds to an item number in the `'DITL'` resource.
- Supplying a window help (`'hwin'`) resource, which will associate help balloons defined in either `'hrct'` resources or `'hdlg'` resources with the dialog's window.

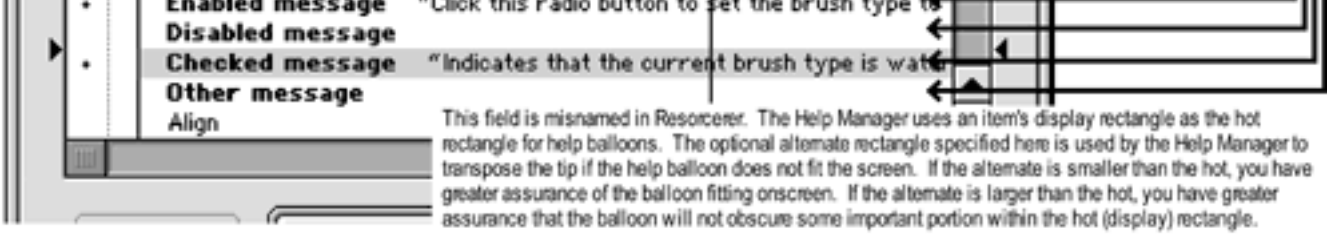
Note

`'hrct'` and `'hwin'` resources are described in Chapter 4.

The option of using a balloon help item (usually referred to as simply a "help item") overcomes the major limitation of the `'hwin'` resource methodology, which is the inability to adequately differentiate between dialogs with no titles (see Chapter 4-Windows, Fig 14). On the other hand, adopting the help item methodology means that you can only associate help balloons with items in the `'DITL'` resource; you cannot provide a single help balloon for a group of related items (unless, of course, they are grouped within a primary or secondary group box).

Figure 14. CREATING A 'hdlg' RESOURCE USING RESORCERER


STRUCTURE OF A COMPILED 'hdlg' RESOURCE
Header component
HELP MANAGER VERSION
INDEX
OPTIONS
BALLOON DEFINITION FUNCTION
VARIATION CODE
ITEM COUNT
MISSING ITEMS COMPONENT
FIRST DIALOG ITEM COMPONENT
...
LAST DIALOG ITEM COMPONENT
Help Manager version
An index into the 'DITL' resource. (See Append 'hdlg', Append help item, and Insert item after:, at Fig 13.)
A number of options. 1 and 4, below, are not relevant to 'hdlg' resources. (2 and 3, below, relate to the three ways that the Help Manager draws and removes balloons.)
Resource ID of the window definition function (WDEF) used for drawing help balloons. The standard WDEF's resource ID is 126. This can be specified by 0 in Resorcerer.
Variation code for WDEF. Governs the location of the balloon's tip.
The number of remaining components defined in the rest of the resource.
Specifies how the Help Manager is to handle items that are not described in this resource. (In the Resorcerer window below, this component has been skipped.)
STRUCTURE OF DIALOG ITEM COMPONENT
SIZE
TYPE OF DATA
TIPS COORDINATES
ALTERNATE RECTANGLE
TEXT STRING
TEXT STRING
TEXT STRING
TEXT STRING
ALIGNMENT BYTES
Pascal string in this component
Coordinates of balloon's tip
Coordinates of alt rectangle
Message for enabled item
Message fo disabled item
Message for item when checked
Other message
The missing items component may be used for purposes similar to the missing item component in 'hmnu' resources. See Fig 13 at Chapter 3 — Menus.
The structure of the hot rectangle component depends on the item chosen in the Message Type pop-up menu in the Resorcerer editing window below, which sets the TYPE OF DATA field. The pop-up menu items specify the format of the help balloon messages. The available formats are as follows:
Use these strings — Use the string specified within this component of this 'hrct' resource. (Specified in this example.)
Use 'PICT' resources — Use the picture stored in the specified 'PICT' resource.
Use 'STR#' resources — Use the specified text string stored in the specified 'STR#' resource.
Used styled text resources — Use the styled text stored in the specified 'TEXT' and 'styl' resources.
Use 'STR ' resources — Use the text string stored in the specified 'STR ' resource.
Skip missing item — No help message. Skip this item.
RESORCERER 'hdlg' RESOURCE EDITING WINDOW
hdlg 130 from MyApp.rsrc
Balloon help version Latest=2
Item offset for first message Start messages at Item #1=0
5-31. Reserved 0
4. For 'hwin's, match string anywhere in title Off
3. Create window, restore bits, and cause update Off
2. Don't create window, restore bits, no update Off
1. Pretend window port origin is set to (0,0) Off
0. Treat resource IDs as sub-IDs for owned resources of
Balloon 'WDEF' Resource ID Standard balloons=0
Balloon variation code (tip position) Along left side at top=0
Dialog items 1
Number of bytes to next record
Missing message type Skip missing item=256
Dialog items #1
Number of bytes to next record
Message type Use these strings=1
Tip (x,y)=(0,0)
Hot rect (t,l,b,r)=(0,0,0,0)
Enabled message "Click this radio button to set the brush type to
Disabled message
Checked message "Indicates that the current brush type is wate
Other message
Align
This field is misnamed in Resorcerer. The Help Manager uses an item's display rectangle as the hot rectangle for help balloons. The optional alternate rectangle specified here is used by the Help Manager to transpose the tip if the help balloon does not fit the screen. If the alternate is smaller than the hot, you have greater assurance of the balloon fitting onscreen. If the alternate is larger than the hot, you have greater assurance that the balloon will not obscure some important portion within the hot (display) rectangle.

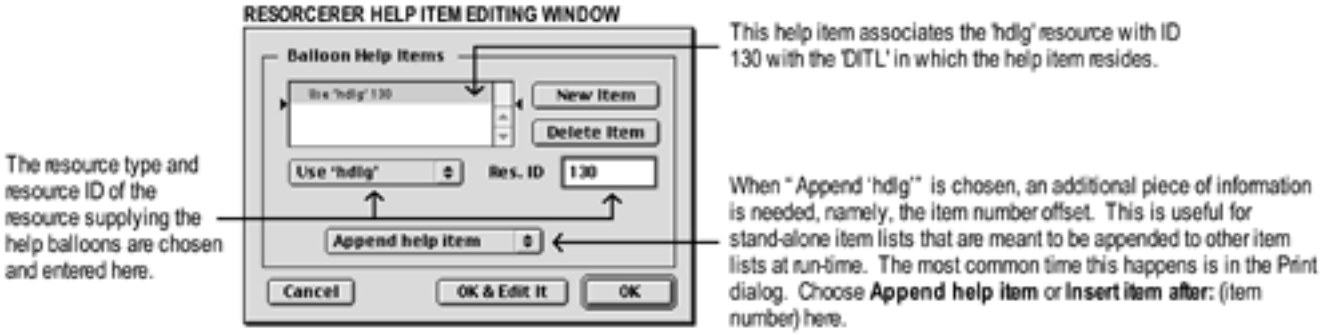

Disabled message
Checked message
Other message
Align
This field is misnamed in Resorcerer. The Help Manager uses an item's display rectangle as the hot rectangle for help balloons. The optional alternate rectangle specified here is used by the Help Manager to transpose the tip if the help balloon does not fit the screen. If the alternate is smaller than the hot, you have greater assurance of the balloon fitting onscreen. If the alternate is larger than the hot, you have greater assurance that the balloon will not obscure some important portion within the hot (display) rectangle.

Help items are invisible. In Resorcerer, the presence of a balloon help item in a `'DITL'` resource is indicated only by a checkmark in the Balloon Help… item in the Item menu. A help item's presence in the `'DITL'` resource is completely ignored by the Dialog Manager.

When the help item methodology is used, the Help Manager automatically tracks the cursor and displays help balloons when the following conditions are met: the dialog has a help item in its `'DITL'` resource; your application calls the Dialog Manager functions `ModalDialog`, or `IsDialogEvent`,; balloon help is enabled.

Figs 13 and 14 in Chapter 4 show `'hrct'` and `'hwin'` resources being created using Resorcerer. Figs 13 and 14 below show a help item and a `'hdlg'` (dialog help) resource being created using Resorcerer.

Figure 13. CREATING A HELP ITEM USING RESORCERER


RESORCERER HELP ITEM EDITING WINDOW
Balloon Help Items
New Item
Delete Item
Use 'hdlg'
Res. ID
130
Append help item
Cancel
OK & Edit It
OK
This help item associates the 'hdlg' resource with ID 130 with the 'DITL' in which the help item resides.
The resource type and resource ID of the resource supplying the help balloons are chosen and entered here.
When "Append 'hdlg'" is chosen, an additional piece of information is needed, namely, the item number offset. This is useful for stand-alone item lists that are meant to be appended to other item lists at run-time. The most common time this happens is in the Print dialog. Choose Append help item or Insert item after: (item number) here.

Help Tags for Dialogs-Mac OS X

Balloon help is not available on Mac OS X. On Mac OS X, you should use help tags instead. Help tag creation is addressed in Chapter 25.

Main Dialog Manager Constants, Data Types and Functions

Constants

Dialog Item Types

```
kControlDialogItem              = 4
kButtonDialogItem               = kControlDialogItem | 0
kCheckBoxDialogItem             = kControlDialogItem | 1
kRadioButtonDialogItem          = kControlDialogItem | 2
kResourceControlDialogItem      = kControlDialogItem | 3
kStaticTextDialogItem           = 8
kEditTextDialogItem             = 16
kIconDialogItem                 = 32
kPictureDialogItem              = 64
kUserDialogItem                 = 0
kItemDisableBit                 = 128
```

Standard Item Numbers for OK and Cancel Push Buttons

```
KStdOKItemIndex                 = 1
KStdCancelItemIndex             = 2
```

Resource IDs of Alert Icons

```
kStopIcon                       = 0
kNoteIcon                       = 1
kCautionIcon                    = 2
```

Dialog Item List Manipulation

```
overlayDITL                     = 0
appendDITLRight                 = 1
appendDITLBottom                = 2
```

Alert Types

```
kAlertStopAlert                 = 0
kAlertNoteAlert                 = 1
kAlertCautionAlert              = 2
kAlertPlainAlert                = 3
```

Standard Alert Push Button Numbers

```
kAlertStdAlertOKButton          = 1
kAlertStdAlertCancelButton      = 2
kAlertStdAlertOtherButton       = 3
kAlertStdAlertHelpButton        = 4
```

Alert Default Text

```
kAlertDefaultOKText             = -1
kAlertDefaultCancelText         = -1
kAlertDefaultOtherText          = -1
```

Dialog Feature Flags

```
kDialogFlagsUseThemeBackground  = (1 << 0)
kDialogFlagsUseControlHierarchy = (1 << 1)
kDialogFlagsHandleMovableModal  = (1 << 2)
kDialogFlagsUseThemeControls    = (1 << 3)
```

Dialog Font Flags

```
kDialogFontNoFontStyle          = 0
kDialogFontUseFontMask          = 0x0001
kDialogFontUseFaceMask          = 0x0002
kDialogFontUseSizeMask          = 0x0004
kDialogFontUseForeColorMask     = 0x0008
kDialogFontUseBackColorMask     = 0x0010
kDialogFontUseModeMask          = 0x0020
kDialogFontUseJustMask          = 0x0040
kDialogFontUseAllMask           = 0x00FF
kDialogFontAddFontSizeMask      = 0x0100
kDialogFontUseFontNameMask      = 0x0200
```

Constants Used in NewFeaturesDialog inProcID Parameter

```
kWindowDocumentProc             = 1024  Modeless dialog
kWindowPlainDialogProc          = 1040  Modal dialog
kWindowShadowDialogProc         = 1041  Modal dialog
kWindowModalDialogProc          = 1042  Modal dialog
kWindowMovableModalDialogProc   = 1043  Movable modal dialog
kWindowAlertProc                = 1044  Modal alert
kWindowMovableAlertProc         = 1045  Movable modal alert
```

Data Types

```
typedef struct OpaqueDialogPtr *DialogPtr;
typedef DialogPtr DialogRef;
```

Standard Alert Parameter Structure

```
struct AlertStdAlertParamRec
{
  Boolean        movable;
  Boolean        helpButton;
  ModalFilterUPP filterProc;
  ConstStringPtr defaultText;
  ConstStringPtr cancelText;
  ConstStringPtr otherText;
  SInt16         defaultButton;
  SInt16         cancelButton;
  UInt16         position;
};
typedef struct AlertStdAlertParamRec AlertStdAlertParamRec;
typedef AlertStdAlertParamRec *AlertStdAlertParamPtr;
```

Standard CFStringAlert Alert Parameter Structure

```
struct AlertStdCFStringAlertParamRec
{
  UInt32      version;
  Boolean     movable;
  Boolean     helpButton;
  CFStringRef defaultText;
  CFStringRef cancelText;
  CFStringRef otherText;
  SInt16      defaultButton;
  SInt16      cancelButton;
  UInt16      position;
  OptionBits  flags;
};
typedef struct AlertStdCFStringAlertParamRec AlertStdCFStringAlertParamRec;
typedef AlertStdCFStringAlertParamRec *AlertStdCFStringAlertParamPtr;
```

Functions

Creating Alerts

```
OSErr      StandardAlert(AlertType inAlertType, ConstStr255Param inError,
           ConstStr255Param inExplanation,const AlertStdAlertParamPtr inAlertParam,
           SInt16 *outItemHit);
OSStatus   CreateStandardAlert(AlertType alertType,CFStringRef error,CFStringRef
explanation,
           const AlertStdCFStringAlertParamRec *param,DialogRef outAlert);
OSStatus   RunStandardAlert(DialogRef inAlert,ModalFilterUPP filterProc,
           DialogItemIndex *outItemHit);
OSStatus   GetStandardAlertDefaultParams(AlertStdCFStringAlertParamPtr param,UInt32
version);
```

Creating, Closing, and Disposing of Dialogs

```
DialogRef  GetNewDialog(short dialogID,void *dStorage,WindowRef behind);
DialogRef  NewFeaturesDialog(void *inStorage, const Rect *inBoundsRect,
           ConstStr255Param inTitle, Boolean inIsVisible,SInt16 inProcID,WindowRef inBehind,
           Boolean inGoAwayFlag,SInt32 inRefCon,Handle inItemListHandle,UInt32 inFlags);
OSErr      AutoSizeDialog(DialogRef inDialog);
void       CloseDialog(DialogRef theDialog);
void       DisposeDialog(DialogRef theDialog);
```

Creating Sheets (Mac OS X Only)

```
OSStatus   CreateStandardSheet(AlertType alertType,CFStringRef  error,CFStringRef
explanation,
           const AlertStdCFStringAlertParamRec *param,EventTargetRef notifyTarget,
           DialogRef *outSheet);
```

Dialog Object Accessor Functions

```
WindowRef  GetDialogWindow(DialogRef dialog);
TEHandle   GetDialogTextEditHandle(DialogRef dialog);
SInt16     GetDialogKeyboardFocusItem(DialogRef dialog);
SInt16     GetDialogDefaultItem(DialogRef dialog);
OSErr      SetDialogDefaultItem(DialogRef theDialog,DialogItemIndex newItem);
SInt16     GetDialogCancelItem(DialogRef dialog);
OSErr      SetDialogCancelItem(DialogRef theDialog,DialogItemIndex newItem);
```

Utility and Casting Functions

```
void       SetPortDialogPort(DialogRef dialog);
CGrafPtr   GetDialogPort(DialogRef dialog);
DialogRef  GetDialogFromWindow(WindowRef window);
```

Manipulating Items in Alerts and Dialogs

```
void       GetDialogItem(DialogRef theDialog,short itemNo,short *itemType,Handle *item,
           Rect *box);
void       SetDialogItem(DialogRef theDialog,short itemNo,short itemType,Handle item,
           const Rect *box);
OSErr      GetDialogItemAsControl(DialogRef inDialog,SInt16 inItemNo,
           ControlHandle *outControl);
OSErr      MoveDialogItem(DialogRef inDialog,SInt16 inItemNo,SInt16 inHoriz,SInt16 inVert);
OSErr      SizeDialogItem(DialogRef inDialog,SInt16 inItemNo,SInt16 inHeight,SInt16
inWidth);
```

```
void       HideDialogItem(DialogRef theDialog,short itemNo);
void       ShowDialogItem(DialogRef theDialog,short itemNo);
short      FindDialogItem(DialogRef theDialog,Point thePt);
void       AppendDITL(DialogRef theDialog,Handle theHandle,DITLMethod theMethod);
void       ShortenDITL(DialogRef theDialog,short numberItems);
OSErr      AppendDialogItemList(DialogRef       dialog,SInt16 ditlID,DITLMethod method);
short      CountDITL(DialogRef the Dialog);
OSStatus   InsertDialogItem (DialogRef theDialog,DialogItemIndex afterItem,
           DialogItemType itemType,Handle itemHandle,const Rect *box);
OSStatus   RemoveDialogItems(DialogRef theDialog,DialogItemIndex itemNo,
           DialogItemIndex amountToRemove,Boolean disposeItemData);
```

Handling Text in Alerts and Dialogs

```
void       ParamText(ConstStr255Param param0,ConstStr255Param param1,ConstStr255Param
param2,
           ConstStr255Param param3);
void       GetParamText(StringPtr param0,StringPtr param1,StringPtr param2,StringPtr
param3);
void       GetDialogItemText(Handle item,Str255 text)
void       SetDialogItemText(Handle item,ConstStr255Param text);
void       SelectDialogItemText(DialogRef theDialog,short itemNo,short strtSel,short
endSel);
void       SetDialogFont(short value);
void       DialogCut(DialogRef theDialog);
void       DialogPaste(DialogRef theDialog);
void       DialogCopy(DialogRef theDialog);
void       DialogDelete(DialogRef theDialog);
```

Handling Events in Dialogs

```
void       ModalDialog(ModalFilterUPP modalFilter,short *itemHit);
Boolean    IsDialogEvent(const EventRecord *theEvent);
Boolean    DialogSelect(const EventRecord *theEvent,DialogRef *theDialog,short *itemHit);
void       UpdateDialog(DialogRef theDialog,RgnHandle updateRgn);
void       DrawDialog(DialogRef theDialog);
OSStatus   SetModalDialogEventMask(DialogRef inDialog,EventMask inMask);
OSStatus   GetModalDialogEventMask(DialogRef inDialog,EventMask *outMask);
OSStatus   SetDialogTimeout(DialogRef inDialog,SInt16 inButtonToPress,UInt32
inSecondsToWait);
OSStatus   GetDialogTimeout(DialogRef inDialog,SInt16 *outButtonToPress,
           UInt32 *outSecondsToWait,UInt32 *outSecondsRemaining);
Boolean    StdFilterProc(DialogRef theDialog,EventRecord *event,DialogItemIndex *itemHit);
OSErr      GetStdFilterProc(ModalFilterUPP *theProc);
OSErr      SetDialogDefaultItem(DialogRef theDialog,DialogItemIndex newItem);
OSErr      SetDialogCancelItem(DialogRef theDialog,DialogItemIndex newItem);
OSErr      SetDialogTracksCursor(DialogRef theDialog,Boolean tracks);
```

Creating and Disposing of Universal Procedure Pointers for Event Filter (Callback) Functions

```
ModalFilterUPP  NewModalFilterUPP(ModalFilterProcPtr userRoutine);
void       DisposeModalFilterUPP(ModalFilterUPP userUPP);
```

Application-Defined (Callback) Function

```
Boolean    myModalFilterFunction(DialogRef theDialog,EventRecord *theEvent,
           DialogItemIndex *itemHit);
```

Relevant Window Manager Functions (Mac OS X Only)

Showing and Hiding Sheets

```
OSStatus   ShowSheetWindow(WindowRef inSheet,WindowRef inParentWindow);
OSStatus   HideSheetWindow(WindowRef inSheet);
OSStatus   GetSheetWindowParent(WindowRef inSheet,WindowRef *outParentWindow);
```

Demonstration Program DialogAndAlerts Listing

```
//
**********************************************************************************************
// DialogsAndAlerts.h                                                    CLASSIC EVENT
MODEL
//
**********************************************************************************************
//
// This program initially opens a small modal dialog which is automatically closed after 10
// seconds, the timeout value having been set by a call to SetDialogTimeout. The program
// then:
//
// •  Opens a window for the purposes of displaying  advisory text and proving correct
window
//    updating and activation/deactivation in the  presence of alerts and dialogs.
//
// •  Allows the user to invoke, via the Demonstration menu, modal and movable modal alerts
//    and dialogs, a modeless dialog and, on Mac OS X, a window-modal alert and dialog
//    (i.e., sheets).
//
// The modal alert box is created programmatically using the StandardAlert function.
//
// The movable modal alert is created programmatically using the StandardAlert function on
Mac
// OS 9 and the CreateStandardAlert function on Mac OS X.
//
// The modal dialog contains three checkboxes in one group box, and two pop-up menu buttons
in
// another group box.
//
// The movable modal dialog contains four radio buttons in one group box, and a clock
control
// and edit text item in another group box.
//
// The modeless dialog contains, amongst other items, an edit text item.
//
// The modal and movable modal alerts and dialogs use an application-defined event filter
// (callback) function.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  An 'MBAR' resource, and 'MENU' resources for Apple, File, and Demonstration pull-down
//    menus, and the pop-up menu buttons (preload, non-purgeable).
//
// •  A 'WIND' resource (purgeable) (initially visible).
//
// •  'DLOG' resources (purgeable) (initially not visible) and associated 'DITL' resources
//    (purgeable), 'dlgx' resources (purgeable), and 'dftb' resources (non-purgeable, but
//    'dftb' resources are automatically marked purgeable when read in).
//
// •  'CNTL' resources for primary group boxes, separator lines, pop-up menu buttons, a
clock,
//    and an image well (all purgeable).
//
// •  'STR#' resources (purgeable) containing the message and informative text for the
alerts.
//
// •  A 'cicn' resource (purgeable) for the modeless dialog box.
//
```

```
// •  A 'ppat' resource (purgeable), which is used to color the content region of the
//    document window for update proving purposes.
//
// •  'hdlg' resources (purgeable) containing balloon help information for the modal and
//    movable modal dialog.
//
// •  An 'hrct' resource and associated 'hwin' resource (both purgeable) containing balloon
//    help information for the modeless dialog.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
//
*********************************************************************************************

// ..............................................................................
includes

#include <Carbon.h>

// ..............................................................................
defines

#define rMenubar                128
#define mAppleApplication       128
#define  iAbout                 1
#define mFile                   129
#define  iClose                 4
#define  iQuit                  12
#define mEdit                   130
#define  iCut                   3
#define  iCopy                  4
#define  iPaste                 5
#define  iClear                 6
#define mDemonstration          131
#define  iModalAlert            1
#define  iMovableAlert          2
#define  iModalDialog           3
#define  iMovableModalDialog    4
#define  iModeless              5
#define  iWindowModalAlert      7
#define  iWindowModalDialog     8
#define mFont                   132
#define rWindow                 128
#define rSplash                 128
#define rModalDialog            129
#define  iGridSnap              4
#define  iShowGrid              5
#define  iShowRulers            6
#define  iFont                  11
#define  iSound                 12
#define rMovableModalDialog     130
#define  iCharcoal              7
#define  iOilPaint              8
#define  iPencil                9
#define  iChalk                 10
#define  iClockOne              12
#define rModelessDialog         131
#define  iEditTextSearchModeless 2
#define rSheetDialog            132
#define  iEditTextSheetDialog   2
#define rAlertStrings           128
#define  sModalMessage          1
#define  sModalInformative      2
#define  sMovableMessage        3
#define  sMovableInformative    4
```

```
#define rSheetStrings           132
#define  sAlertSheetMessage     1
#define  sAlertSheetInformative 2
#define rPixelPattern           128
#define kSearchModeless         1
#define kSheetDialog            2

#define kReturn                 (SInt8) 0x0D
#define kEnter                  (SInt8) 0x03
#define kEscape                 (SInt8) 0x1B
#define kPeriod                 (SInt8) 0x2E

#define MAX_UINT32              0xFFFFFFFF

// ..................................................................... function
prototypes

void    main                            (void);
void    doPreliminaries                 (void);
OSErr   quitAppEventHandler             (AppleEvent *,AppleEvent *,SInt32);
void    eventLoop                       (void);
void    doIdle                          (void);
void    doEvents                        (EventRecord *);
void    doMouseDown                     (EventRecord *);
void    doKeyDown                       (EventRecord *);
void    doUpdate                        (EventRecord *);
void    doUpdateDocument                (WindowRef);
void    doActivate                      (EventRecord *);
void    doActivateDocument              (WindowRef,Boolean);
void    doActivateDialogs               (EventRecord *,Boolean);
void    doOSEvent                       (EventRecord *);
void    doAdjustMenus                   (void);
void    doMenuChoice                    (SInt32);
void    doEditMenu                      (MenuItemIndex);
void    doDemonstrationMenu             (MenuItemIndex);
void    doExplicitlyDeactivateDocument (void);
Boolean doModalAlerts                   (Boolean);
Boolean doMovableModalAlertOnX          (void);
Boolean doModalDialog                   (void);
Boolean doMovableModalDialog            (void);
Boolean doCreateOrShowModelessDialog    (void);
void    doInContent                     (EventRecord *);
void    doButtonHitInSearchModeless     (void);
void    doHideModelessDialog            (WindowRef);
Boolean eventFilter                     (DialogRef,EventRecord *,SInt16 *);
void    doPopupMenuChoice               (ControlRef,SInt16);
void    doDrawMessage                   (WindowRef,Boolean);
void    doCopyPString                   (Str255,Str255);

Boolean doSheetAlert                    (void);
Boolean doSheetDialog                   (void);
void    doButtonHitInSheetDialog        (void);

void    helpTagsModal                   (DialogRef);
void    helpTagsMovableModal            (DialogRef);
void    helpTagsModeless                (DialogRef);

//
******************************************************************************************
// DialogsAndAlerts.c
//
******************************************************************************************

// ..........................................................................
includes
```

```
#include "DialogsAndAlerts.h"

// ..................................................................... global
variables

Boolean        gRunningOnX        = false;
ModalFilterUPP gEventFilterUPP;
Str255         gCurrentString;
WindowRef      gWindowRef;
SInt32         gSleepTime;
Boolean        gDone;
Boolean        gGridSnap          = kControlCheckBoxUncheckedValue;
Boolean        gShowGrid          = kControlCheckBoxUncheckedValue;
Boolean        gShowRule          = kControlCheckBoxUncheckedValue;
SInt16         gBrushType         = iCharcoal;
DialogRef      gModelessDialogRef = NULL;

// ********************************************************************************
main

void  main(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  DialogRef     dialogRef;
  SInt16        itemHit;

  // ..................................................................... do
preliminaries

  doPreliminaries();

  // ............................................................... set up menu bar and
menus

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    ExitToShell();
  SetMenuBar(menubarHdl);
  DrawMenuBar();

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
      DisableMenuItem(menuRef,0);
    }

    gRunningOnX = true;
  }

  // ................... open small modal dialog and automatically dismiss it after 10
seconds

  dialogRef = GetNewDialog(rSplash,NULL,(WindowRef) -1);
  SetDialogTimeout(dialogRef,kStdOkItemIndex,10);

  do
  {
    ModalDialog(NULL,&itemHit);
  }  while(itemHit != kStdOkItemIndex);
```

```
  DisposeDialog(dialogRef);

  // ........................... create universal procedure pointer for event filter
function

  gEventFilterUPP = NewModalFilterUPP((ModalFilterProcPtr) eventFilter);

  // ............................................... initial advisory text for window
header

  doCopyPString("\pBalloon (OS 8/9) and Help tag (OS X) help is available",gCurrentString);

  // ..................................................... open a window, set font
size

  if(!(gWindowRef = GetNewCWindow(rWindow,NULL,(WindowRef)-1)))
    ExitToShell();
  SetPortWindowPort(gWindowRef);
  if(!gRunningOnX)
    TextSize(10);

  // ..................................................................... enter
eventLoop

  eventLoop();
}

// ************************************************************************
doPreliminaries

void  doPreliminaries(void)
{
  OSErr osError;

  MoreMasterPointers(192);
  InitCursor();
  FlushEvents(everyEvent,0);

  osError = AEInstallEventHandler(kCoreEventClass,kAEQuitApplication,
                           NewAEEventHandlerUPP((AEEventHandlerProcPtr)
quitAppEventHandler),
                           0L,false);
  if(osError != noErr)
    ExitToShell();
}

// ************************************************************************
doQuitAppEvent

OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  OSErr    osError;
  DescType returnedType;
  Size     actualSize;

  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,
NULL,0,
                              &actualSize);

  if(osError == errAEDescNotFound)
  {
    gDone = true;
    osError = noErr;
  }
  else if(osError == noErr)
```

```
    osError = errAEParamMissed;

  return osError;
}

// ********************************************************************************
eventLoop

void  eventLoop(void)
{
  EventRecord eventStructure;
  Boolean     gotEvent;
  gSleepTime = MAX_UINT32;
  gDone = false;

  while(!gDone)
  {
    gotEvent = WaitNextEvent(everyEvent,&eventStructure,gSleepTime,NULL);
    if(gotEvent)
      doEvents(&eventStructure);
    else
    {
      if(eventStructure.what == nullEvent)
        if(!gRunningOnX)
          doIdle();
    }
  }
}

// ********************************************************************************
doIdle

void  doIdle(void)
{
  if(FrontWindow() == GetDialogWindow(gModelessDialogRef))
    IdleControls(GetDialogWindow(gModelessDialogRef));
}

// ********************************************************************************
doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
  switch(eventStrucPtr->what)
  {
    case kHighLevelEvent:
      AEProcessAppleEvent(eventStrucPtr);
      break;

    case mouseDown:
      doMouseDown(eventStrucPtr);
      break;

    case keyDown:
      doKeyDown(eventStrucPtr);
      break;

    case autoKey:
      if((eventStrucPtr->modifiers & cmdKey) == 0)
        doKeyDown(eventStrucPtr);
      break;

    case updateEvt:
      doUpdate(eventStrucPtr);
      break;
```

```
    case activateEvt:
      doActivate(eventStrucPtr);
      break;
    case osEvt:
      doOSEvent(eventStrucPtr);
      break;
  }
}

// ****************************************************************************
doMouseDown

void  doMouseDown(EventRecord *eventStrucPtr)
{
  WindowRef      windowRef;
  WindowPartCode partCode;

  partCode = FindWindow(eventStrucPtr->where,&windowRef);

  switch(partCode)
  {
    case inMenuBar:
      doAdjustMenus();
      doMenuChoice(MenuSelect(eventStrucPtr->where));
      break;

    case inContent:
      if(windowRef != FrontWindow())
        SelectWindow(windowRef);
      else
        doInContent(eventStrucPtr);
      break;

    case inDrag:
      DragWindow(windowRef,eventStrucPtr->where,NULL);
      break;

    case inGoAway:
      if(TrackGoAway(windowRef,eventStrucPtr->where))
      {
        if(GetWindowKind(windowRef) == kDialogWindowKind)
        {
          doHideModelessDialog(windowRef);
          doCopyPString("\pBalloon (OS 8/9) and Help tag (OS X) help is available",
                        gCurrentString);
        }
      }
      break;
  }
}

// ****************************************************************************
doKeyDown

void  doKeyDown(EventRecord *eventStrucPtr)
{
  WindowRef  windowRef;
  SInt8      charCode;
  SInt32     windowRefCon;
  SInt16     itemHit;
  ControlRef controlRef;
  UInt32     finalTicks;
  DialogRef  dialogRef;

  windowRef = FrontWindow();
  charCode = eventStrucPtr->message & charCodeMask;
```

```
  if(!(IsDialogEvent(eventStrucPtr)))
  {
    if((eventStrucPtr->modifiers & cmdKey) != 0)
    {
      doAdjustMenus();
      doMenuChoice(MenuEvent(eventStrucPtr));
    }
  }
  else
  {
    windowRefCon = GetWRefCon(windowRef);
    if(windowRefCon == kSearchModeless || windowRefCon == kSheetDialog)
    {
      if((charCode == kReturn) || (charCode == kEnter))
      {
        GetDialogItemAsControl(GetDialogFromWindow(windowRef),kStdOkItemIndex,&controlRef);
        HiliteControl(controlRef,kControlButtonPart);
        Delay(8,&finalTicks);
        HiliteControl(controlRef,kControlEntireControl);
        if(windowRefCon == kSearchModeless)
          doButtonHitInSearchModeless();
        else if(windowRefCon == kSheetDialog)
          doButtonHitInSheetDialog();
        return;
      }

      if((eventStrucPtr->modifiers & cmdKey) != 0)
      {
        if(charCode == 'X' || charCode == 'x' ||  charCode == 'C' || charCode == 'c' ||
            charCode == 'V' || charCode == 'v')
        {
          HiliteMenu(mEdit);
          DialogSelect(eventStrucPtr,&dialogRef,&itemHit);
          Delay(4,&finalTicks);
          HiliteMenu(0);
        }
        else
        {
          doAdjustMenus();
          doMenuChoice(MenuEvent(eventStrucPtr));
        }

        return;
      }

      DialogSelect(eventStrucPtr,&dialogRef,&itemHit);
    }
  }
}
// ****************************************************************************
doUpdate

void  doUpdate(EventRecord *eventStrucPtr)
{
  WindowRef windowRef;
  DialogRef dialogRef;
  SInt16    itemHit;

  if(!(IsDialogEvent(eventStrucPtr)))
  {
    windowRef = (WindowRef) eventStrucPtr->message;
    doUpdateDocument(windowRef);
  }
  else
    DialogSelect(eventStrucPtr,&dialogRef,&itemHit);
```

```
}

// ****************************************************************************
doUpdateDocument

void  doUpdateDocument(WindowRef windowRef)
{
  GrafPtr      oldPort;
  PixPatHandle pixpatHdl;
  Rect         portRect;

  BeginUpdate(windowRef);

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

  pixpatHdl = GetPixPat(rPixelPattern);
  GetWindowPortBounds(windowRef,&portRect);
  FillCRect(&portRect,pixpatHdl);
  DisposePixPat(pixpatHdl);
  doDrawMessage(windowRef,windowRef == FrontWindow());

  SetPort(oldPort);

  EndUpdate(windowRef);
}

// ****************************************************************************
doActivate

void  doActivate(EventRecord *eventStrucPtr)
{
  Boolean   becomingActive;
  WindowRef windowRef;

  becomingActive = (eventStrucPtr->modifiers & activeFlag) == activeFlag;

  if(!(IsDialogEvent(eventStrucPtr)))
  {
    windowRef = (WindowRef) eventStrucPtr->message;
    doActivateDocument(windowRef,becomingActive);
  }
  else
    doActivateDialogs(eventStrucPtr,becomingActive);
}

// ****************************************************************************
doActivateDocument

void  doActivateDocument(WindowRef windowRef,Boolean becomingActive)
{
  if(becomingActive)
    doAdjustMenus();

  doDrawMessage(windowRef,becomingActive);
}

// ****************************************************************************
doActivateModelessDialog

void  doActivateDialogs(EventRecord *eventStrucPtr,Boolean becomingActive)
{
  DialogRef dialogRef;
  SInt16    windowRefCon;
  SInt16    itemHit;
```

```
  DialogSelect(eventStrucPtr,&dialogRef,&itemHit);

  windowRefCon = GetWRefCon(GetDialogWindow(dialogRef));

  if(becomingActive)
  {
    doAdjustMenus();
    if(windowRefCon == kSearchModeless || windowRefCon == kSheetDialog)
      gSleepTime = GetCaretTime();
  }
  else
  {
    if(windowRefCon == kSearchModeless || windowRefCon == kSheetDialog)
      gSleepTime = MAX_UINT32;
  }
}

// ********************************************************************************
doOSEvent

void  doOSEvent(EventRecord *eventStrucPtr)
{
  switch((eventStrucPtr->message >> 24) & 0x000000FF)
  {
    case suspendResumeMessage:
      if((eventStrucPtr->message & resumeFlag) == 1)
        SetThemeCursor(kThemeArrowCursor);
      break;
  }
}
// ********************************************************************************
doAdjustMenus

void  doAdjustMenus(void)
{
  WindowRef windowRef;
  MenuRef   menuRef;
  SInt32    windowRefCon;

  windowRef = FrontWindow();

  if(GetWindowKind(windowRef) == kApplicationWindowKind)
  {
    menuRef = GetMenuRef(mFile);
    DisableMenuItem(menuRef,iClose);
    menuRef = GetMenuRef(mEdit);
    DisableMenuItem(menuRef,0);
    menuRef = GetMenuRef(mDemonstration);
    EnableMenuItem(menuRef,iModeless);
    if(gRunningOnX)
    {
      if(IsWindowCollapsed(gWindowRef))
      {
        DisableMenuItem(menuRef,iWindowModalDialog);
        DisableMenuItem(menuRef,iWindowModalAlert);
      }
      else
      {
        EnableMenuItem(menuRef,iWindowModalDialog);
        EnableMenuItem(menuRef,iWindowModalAlert);
      }
    }
  }
  else if(GetWindowKind(windowRef) == kDialogWindowKind)
  {
    windowRefCon = GetWRefCon(windowRef);
```

```
    if(windowRefCon == kSearchModeless)
    {
      menuRef = GetMenuRef(mFile);
      EnableMenuItem(menuRef,iClose);
      menuRef = GetMenuRef(mEdit);
      EnableMenuItem(menuRef,0);
      menuRef = GetMenuRef(mDemonstration);
      DisableMenuItem(menuRef,iModeless);
    }
    else if(windowRefCon == kSheetDialog)
    {
      menuRef = GetMenuRef(mFile);
      DisableMenuItem(menuRef,iClose);
      menuRef = GetMenuRef(mEdit);
      EnableMenuItem(menuRef,0);
      menuRef = GetMenuRef(mDemonstration);
      EnableMenuItem(menuRef,iModeless);
      DisableMenuItem(menuRef,iWindowModalAlert);
    }
    else
    {
      menuRef = GetMenuRef(mFile);
      DisableMenuItem(menuRef,iClose);
      menuRef = GetMenuRef(mEdit);
      DisableMenuItem(menuRef,0);
      menuRef = GetMenuRef(mDemonstration);
      EnableMenuItem(menuRef,iModeless);
    }
  }
  DrawMenuBar();
}

// ****************************************************************************
doMenuChoice

void  doMenuChoice(SInt32 menuChoice)
{
  MenuID        menuID;
  MenuItemIndex menuItem;
  SInt16        windowRefCon;

  menuID = HiWord(menuChoice);
  menuItem = LoWord(menuChoice);

  if(menuID == 0)
    return;

  switch(menuID)
  {
    case mAppleApplication:
      if(menuItem == iAbout)
        SysBeep(10);
      break;

    case mFile:
      if(menuItem == iQuit)
        gDone = true;
      else if(menuItem == iClose)
      {
        if(GetWindowKind(FrontWindow()) == kDialogWindowKind)
        {
          windowRefCon = GetWRefCon(FrontWindow());
          if(windowRefCon == kSearchModeless)
            doHideModelessDialog(GetDialogWindow(gModelessDialogRef));
        }
      }
```

```
      break;

    case mEdit:
      doEditMenu(menuItem);
      break;
    case mDemonstration:
      doDemonstrationMenu(menuItem);
      break;
  }

  HiliteMenu(0);
}

// *****************************************************************************
doEditMenu

void  doEditMenu(MenuItemIndex menuItem)
{
  WindowRef windowRef;
  SInt16    windowRefCon;
  DialogRef dialogRef;

  windowRef = FrontWindow();

  if(GetWindowKind(FrontWindow()) == kDialogWindowKind)
  {
    windowRefCon = GetWRefCon(windowRef);
    if(windowRefCon == kSearchModeless || windowRefCon == kSheetDialog)
    {
      dialogRef = GetDialogFromWindow(windowRef);

      switch(menuItem)
      {
        case iCut:
          DialogCut(dialogRef);
          break;

        case iCopy:
          DialogCopy(dialogRef);
          break;

        case iPaste:
          DialogPaste(dialogRef);
          break;

        case iClear:
          DialogDelete(dialogRef);
          break;
      }
    }
  }
}

// *****************************************************************************
doDemonstrationMenu

void  doDemonstrationMenu(MenuItemIndex menuItem)
{
  switch(menuItem)
  {
    case iModalAlert:
      if(!doModalAlerts(false))
      {
        SysBeep(10);
        ExitToShell();
      }
```

```
      break;

    case iMovableAlert:
      if(gRunningOnX)
      {
        if(!doMovableModalAlertOnX())
        {
          SysBeep(10);
          ExitToShell();
        }
      }
      else if(!doModalAlerts(true))
      {
        SysBeep(10);
        ExitToShell();
      }
      break;

    case iModalDialog:
      if(!doModalDialog())
      {
        SysBeep(10);
        ExitToShell();
      }
      break;

    case iMovableModalDialog:
      if(!doMovableModalDialog())
      {
        SysBeep(10);
        ExitToShell();
      }
      break;

    case iModeless:
      if(!doCreateOrShowModelessDialog())
      {
        SysBeep(10);
        ExitToShell();
      }
      break;

    case iWindowModalAlert:
      if(!doSheetAlert())
      {
        SysBeep(10);
        ExitToShell();
      }
      break;
    case iWindowModalDialog:
      if(!doSheetDialog())
      {
        SysBeep(10);
        ExitToShell();
      }
      break;
  }
}

// ************************************************************
doExplicitlyDeactivateDocument

void  doExplicitlyDeactivateDocument(void)
{
  if(FrontWindow() && (GetWindowKind(FrontWindow()) != kDialogWindowKind))
    doActivateDocument(FrontWindow(),false);
```

```
}

// ************************************************************************
doModalAlerts

Boolean  doModalAlerts(Boolean movable)
{
  AlertStdAlertParamRec paramRec;
  Str255                messageText, informativeText;
  Str255                otherText = "\pOther";
  OSErr                 osError;
  DialogItemIndex       itemHit;

  doExplicitlyDeactivateDocument();

  paramRec.movable       = movable;
  paramRec.helpButton    = true;
  paramRec.filterProc    = gEventFilterUPP;
  paramRec.defaultText   = (StringPtr) kAlertDefaultOKText;
  paramRec.cancelText    = (StringPtr) kAlertDefaultCancelText;
  paramRec.otherText     = (StringPtr) &otherText;
  paramRec.defaultButton = kAlertStdAlertOKButton;
  paramRec.cancelButton  = kAlertStdAlertCancelButton;
  paramRec.position      = kWindowDefaultPosition;

  if(!movable)
    GetIndString(messageText,rAlertStrings,sModalMessage);
  else
    GetIndString(messageText,rAlertStrings,sMovableMessage);
  GetIndString(informativeText,rAlertStrings,sModalInformative);

  osError = StandardAlert(kAlertStopAlert,messageText,informativeText,&paramRec,&itemHit);
  if(osError == noErr)
  {
    if(itemHit == kAlertStdAlertOKButton)
      doCopyPString("\pOK Button hit",gCurrentString);
    else if (itemHit == kAlertStdAlertCancelButton)
      doCopyPString("\pCancel Button hit",gCurrentString);
    else if (itemHit == kAlertStdAlertOtherButton)
      doCopyPString("\pOther Button hit",gCurrentString);
    else if (itemHit == kAlertStdAlertHelpButton)
      doCopyPString("\pHelp Button hit",gCurrentString);
  }

  return (osError == noErr);
}

// ************************************************************************
doMovableModalAlertOnX

Boolean  doMovableModalAlertOnX(void)
{
  AlertStdCFStringAlertParamRec paramRec;
  Str255                        messageText, informativeText;
  CFStringRef                   messageTextCF, informativeTextCF;
  OSErr                         osError;
  DialogRef                     dialogRef;
  DialogItemIndex               itemHit;

  doExplicitlyDeactivateDocument();

  GetStandardAlertDefaultParams(&paramRec,kStdCFStringAlertVersionOne);
  paramRec.movable      = true;
  paramRec.helpButton   = true;
  paramRec.cancelButton = kAlertStdAlertCancelButton;
  paramRec.cancelText   = CFSTR("Cancel");
```

```
  paramRec.otherText    = CFSTR("Other");

  GetIndString(messageText,rAlertStrings,sMovableMessage);
  GetIndString(informativeText,rAlertStrings,sMovableInformative);
  messageTextCF = CFStringCreateWithPascalString(NULL,messageText,
                                                 CFStringGetSystemEncoding());
  informativeTextCF = CFStringCreateWithPascalString(NULL,informativeText,
                                                     CFStringGetSystemEncoding());

  osError = CreateStandardAlert(kAlertCautionAlert,messageTextCF,informativeTextCF,
&paramRec,
                                &dialogRef);
  if(osError == noErr)
  {
    osError = RunStandardAlert(dialogRef,NULL,&itemHit);
    if(osError == noErr)
    {
      if(itemHit == kAlertStdAlertOKButton)
        doCopyPString("\pOK Button hit",gCurrentString);
      else if (itemHit == kAlertStdAlertCancelButton)
        doCopyPString("\pCancel Button hit",gCurrentString);
      else if (itemHit == kAlertStdAlertOtherButton)
        doCopyPString("\pOther Button hit",gCurrentString);
      else if (itemHit == kAlertStdAlertHelpButton)
        doCopyPString("\pHelp Button hit",gCurrentString);
    }
  }
  if(messageTextCF != NULL)
    CFRelease(messageTextCF);
  if(informativeTextCF != NULL)
    CFRelease(informativeTextCF);

  return (osError == noErr);
}

// ****************************************************************************
doModalDialog

Boolean  doModalDialog(void)
{
  DialogRef  dialogRef;
  ControlRef controlRef;
  OSStatus   osError;
  MenuRef    menuRef;
  SInt16     numberOfItems, itemHit, controlValue;

  doExplicitlyDeactivateDocument();

  if(!(dialogRef = GetNewDialog(rModalDialog,NULL,(WindowRef) -1)))
    return false;

  SetDialogDefaultItem(dialogRef,kStdOkItemIndex);
  SetDialogCancelItem(dialogRef,kStdCancelItemIndex);

  GetDialogItemAsControl(dialogRef,iGridSnap,&controlRef);
  SetControlValue(controlRef,gGridSnap);
  GetDialogItemAsControl(dialogRef,iShowGrid,&controlRef);
  SetControlValue(controlRef,gShowGrid);
  GetDialogItemAsControl(dialogRef,iShowRulers,&controlRef);
  SetControlValue(controlRef,gShowRule);

  menuRef = NewMenu(mFont,NULL);

  if((osError = CreateStandardFontMenu(menuRef,0,0,0,NULL)) == noErr)
  {
    GetDialogItemAsControl(dialogRef,iFont,&controlRef);
```

```
    SetControlMinimum(controlRef,1);
    numberOfItems = CountMenuItems(menuRef);
    SetControlMaximum(controlRef,numberOfItems);
    SetControlData(controlRef,kControlEntireControl,kControlPopupButtonMenuRefTag,
                   sizeof(menuRef),&menuRef);
  }
  else
    return false;

  if(gRunningOnX)
    helpTagsModal(dialogRef);

  ShowWindow(GetDialogWindow(dialogRef));

  do
  {
    ModalDialog(gEventFilterUPP,&itemHit);
    if(itemHit == iGridSnap || itemHit == iShowGrid || itemHit == iShowRulers)
    {
      GetDialogItemAsControl(dialogRef,itemHit,&controlRef);
      SetControlValue(controlRef,!GetControlValue(controlRef));
    }
    else if(itemHit == iFont || itemHit == iSound)
    {
      GetDialogItemAsControl(dialogRef,itemHit,&controlRef);
      controlValue = GetControlValue(controlRef);
      doPopupMenuChoice(controlRef,controlValue);
    }
  }  while((itemHit != kStdOkItemIndex) && (itemHit != kStdCancelItemIndex));

  if(itemHit == kStdOkItemIndex)
  {
    GetDialogItemAsControl(dialogRef,iGridSnap,&controlRef);
    gGridSnap = GetControlValue(controlRef);
    GetDialogItemAsControl(dialogRef,iShowGrid,&controlRef);
    gShowGrid = GetControlValue(controlRef);
    GetDialogItemAsControl(dialogRef,iShowRulers,&controlRef);
    gShowRule = GetControlValue(controlRef);
  }

  DisposeDialog(dialogRef);

  doCopyPString("\pBalloon (OS 8/9) and Help tag (OS X) help is available",gCurrentString);

  return true;
}

// *********************************************************************
doMovableModalDialog

Boolean  doMovableModalDialog(void)
{
  DialogRef  dialogRef;
  ControlRef controlRef;
  SInt16     oldBrushType, itemHit, a;

  doExplicitlyDeactivateDocument();

  if(!(dialogRef = GetNewDialog(rMovableModalDialog,NULL,(WindowRef) -1)))
    return false;

  SetDialogDefaultItem(dialogRef,kStdOkItemIndex);
  SetDialogCancelItem(dialogRef,kStdCancelItemIndex);
  SetDialogTracksCursor(dialogRef,true);

  GetDialogItemAsControl(dialogRef,gBrushType,&controlRef);
```

```
  SetControlValue(controlRef,kControlRadioButtonCheckedValue);

  GetDialogItemAsControl(dialogRef,iClockOne,&controlRef);
  SetKeyboardFocus(GetDialogWindow(dialogRef),controlRef,kControlClockPart);

  oldBrushType = gBrushType;
  if(gRunningOnX)
    helpTagsMovableModal(dialogRef);

  ShowWindow(GetDialogWindow(dialogRef));

  do
  {
    ModalDialog(gEventFilterUPP,&itemHit);

    if(itemHit >= iCharcoal && itemHit <= iChalk)
    {
      for(a=iCharcoal;a<=iChalk;a++)
      {
        GetDialogItemAsControl(dialogRef,a,&controlRef);
        SetControlValue(controlRef,kControlRadioButtonUncheckedValue);
      }

      GetDialogItemAsControl(dialogRef,itemHit,&controlRef);
      SetControlValue(controlRef,kControlRadioButtonCheckedValue);
      gBrushType = itemHit;
    }
  }  while((itemHit != kStdOkItemIndex) && (itemHit != kStdCancelItemIndex));

  if(itemHit == kStdCancelItemIndex)
    gBrushType = oldBrushType;

  DisposeDialog(dialogRef);

  return true;
}

// ************************************************************
doCreateOrShowModelessDialog

Boolean  doCreateOrShowModelessDialog(void)
{
  ControlRef controlRef;
  Str255     stringData = "\pwicked googly";
  MenuRef    menuRef;

  if(gModelessDialogRef == NULL)
  {
    if(!(gModelessDialogRef = GetNewDialog(rModelessDialog,NULL,(WindowRef) -1)))
      return false;

    SetWRefCon(GetDialogWindow(gModelessDialogRef),(SInt32) kSearchModeless);

    SetDialogDefaultItem(gModelessDialogRef,kStdOkItemIndex);

    GetDialogItemAsControl(gModelessDialogRef,iEditTextSearchModeless,&controlRef);
    SetDialogItemText((Handle) controlRef,stringData);
    SelectDialogItemText(gModelessDialogRef,iEditTextSearchModeless,0,32767);

    if(gRunningOnX)
      helpTagsModeless(gModelessDialogRef);
    ShowWindow(GetDialogWindow(gModelessDialogRef));
  }
  else
  {
    ShowWindow(GetDialogWindow(gModelessDialogRef));
```

```
    SelectWindow(GetDialogWindow(gModelessDialogRef));
  }

  if(gRunningOnX)
  {
    menuRef = GetMenuRef(mFile);
    EnableMenuItem(menuRef,0);
  }

  return true;
}

// ****************************************************************************
doInContent

void  doInContent(EventRecord *eventStrucPtr)
{
  WindowRef windowRef;
  SInt32    windowRefCon;
  DialogRef dialogRef;
  SInt16    itemHit;

  windowRef = FrontWindow();

  if(!(IsDialogEvent(eventStrucPtr)))
  {
    // Handle clicks in document window content region here.
  }
  else
  {
    windowRefCon = GetWRefCon(windowRef);
    if(windowRefCon == kSearchModeless)
    {
      if(DialogSelect(eventStrucPtr,&dialogRef,&itemHit))
        if(itemHit == kStdOkItemIndex)
          doButtonHitInSearchModeless();
    }
    else if(windowRefCon == kSheetDialog)
    {
      if(DialogSelect(eventStrucPtr,&dialogRef,&itemHit))
        if(itemHit == kStdOkItemIndex)
          doButtonHitInSheetDialog();
    }
  }
}

// ************************************************************
doButtonHitInSearchModeless

void  doButtonHitInSearchModeless(void)
{
  ControlRef controlRef;
  GrafPtr    oldPort;

  GetDialogItemAsControl(gModelessDialogRef,iEditTextSearchModeless,&controlRef);
  GetDialogItemText((Handle) controlRef,gCurrentString);

  GetPort(&oldPort);
  SetPortWindowPort(gWindowRef);
  doDrawMessage(gWindowRef,false);
  SetPort(oldPort);
}

// ********************************************************************
doHideModelessDialog
```

```
void  doHideModelessDialog(WindowRef windowRef)
{
  SInt16  windowRefCon;
  MenuRef menuRef;

  if(gRunningOnX)
    BringToFront(gWindowRef);

  HideWindow(windowRef);

  windowRefCon = GetWRefCon(windowRef);
  if(windowRefCon == kSearchModeless)
    gSleepTime = MAX_UINT32;

  if(gRunningOnX)
  {
    menuRef = GetMenuRef(mFile);
    DisableMenuItem(menuRef,0);
  }
}

// ****************************************************************************
eventFilter

Boolean  eventFilter(DialogRef dialogRef,EventRecord *eventStrucPtr,SInt16 *itemHit)
{
  Boolean handledEvent;
  GrafPtr oldPort;

  handledEvent = false;

  if((eventStrucPtr->what == updateEvt) &&
     ((WindowRef) eventStrucPtr->message != GetDialogWindow(dialogRef)))
  {
    if(!gRunningOnX)
      doUpdate(eventStrucPtr);
  }
  else if((eventStrucPtr->what == autoKey) && ((eventStrucPtr->modifiers & cmdKey) != 0))
  {
    handledEvent = true;
    return handledEvent;
  }
  else
  {
    GetPort(&oldPort);
    SetPortDialogPort(dialogRef);

    handledEvent = StdFilterProc(dialogRef,eventStrucPtr,itemHit);

    SetPort(oldPort);
  }

  return handledEvent;
}

// ****************************************************************************
doPopupMenuChoice

void  doPopupMenuChoice(ControlRef controlRef,SInt16 controlValue)
{
  MenuRef menuRef;
  Size    actualSize;
  Str255  itemName;
  GrafPtr oldPort;

  GetControlData(controlRef,kControlEntireControl,kControlPopupButtonMenuHandleTag,
```

```
                sizeof(menuRef),&menuRef,&actualSize);
  GetMenuItemText(menuRef,controlValue,itemName);
  doCopyPString(itemName,gCurrentString);

  GetPort(&oldPort);
  SetPortWindowPort(gWindowRef);
  doDrawMessage(gWindowRef,false);
  SetPort(oldPort);
}

// *****************************************************************************
doDrawMessage

void  doDrawMessage(WindowRef windowRef,Boolean inState)
{
  Rect        portRect, headerRect;
  CFStringRef stringRef;
  Rect        textBoxRect;

  if(windowRef == gWindowRef)
  {
    SetPortWindowPort(windowRef);
    GetWindowPortBounds(windowRef,&portRect);
    SetRect(&headerRect,portRect.left - 1,portRect.bottom - 26,portRect.right + 1,
            portRect.bottom + 1);
    DrawThemePlacard(&headerRect,inState);

    if(inState == kThemeStateActive)
      TextMode(srcOr);
    else
      TextMode(grayishTextOr);
    stringRef = CFStringCreateWithPascalString(NULL,gCurrentString,
                                               CFStringGetSystemEncoding());
    SetRect(&textBoxRect,portRect.left,portRect.bottom - 19,portRect.right,
              portRect.bottom - 4);
    DrawThemeTextBox(stringRef,kThemeSmallSystemFont,0,true,&textBoxRect,teJustCenter,NULL);
    if(stringRef != NULL)
      CFRelease(stringRef);

    TextMode(srcOr);
  }
}

// *****************************************************************************
doCopyPString

void  doCopyPString(Str255 sourceString,Str255 destinationString)
{
  SInt16 stringLength;

  stringLength = sourceString[0];
  BlockMove(sourceString + 1,destinationString + 1,stringLength);
  destinationString[0] = stringLength;
}

//
*******************************************************************************************
// Sheets.c
//
*******************************************************************************************

// .....................................................................
includes

#include "DialogsAndAlerts.h"
```

```
// ..................................................................... global
variables

WindowRef gSheetDialogWindowRef = NULL;

extern WindowRef gWindowRef;
extern Boolean   gRunningOnX;
extern Str255    gCurrentString;

// ***************************************************************************
doSheetAlert

Boolean  doSheetAlert(void)
{
  AlertStdCFStringAlertParamRec paramRec;
  Str255                        messageText, informativeText;
  CFStringRef                   messageTextCF, informativeTextCF;
  OSStatus                      osError;
  DialogRef                     dialogRef;
  MenuRef                       menuRef;

  GetStandardAlertDefaultParams(&paramRec,kStdCFStringAlertVersionOne);

  GetIndString(messageText,rSheetStrings,sAlertSheetMessage);
  GetIndString(informativeText,rSheetStrings,sAlertSheetInformative);
  messageTextCF = CFStringCreateWithPascalString(NULL,messageText,
                                                 CFStringGetSystemEncoding());
  informativeTextCF = CFStringCreateWithPascalString(NULL,informativeText,
                                                     CFStringGetSystemEncoding());

  osError = CreateStandardSheet(kAlertCautionAlert,messageTextCF,informativeTextCF,
&paramRec,
                                GetWindowEventTarget(gWindowRef),&dialogRef);
  if(osError == noErr)
    osError = ShowSheetWindow(GetDialogWindow(dialogRef),gWindowRef);

  CFRelease(messageTextCF);
  CFRelease(informativeTextCF);

  menuRef = GetMenuRef(mDemonstration);
  if(menuRef != NULL)
  {
    DisableMenuItem(menuRef,iWindowModalDialog);
    DisableMenuItem(menuRef,iWindowModalAlert);
  }

  return (osError == noErr);
}

// ***************************************************************************
doSheetDialog

Boolean  doSheetDialog(void)
{
  DialogRef  dialogRef;
  ControlRef controlRef;
  Str255     stringData = "\pBradman";
  OSStatus   osError = noErr;
  MenuRef    menuRef;

  if(!(dialogRef = GetNewDialog(rSheetDialog,NULL,(WindowRef) -1)))
    return false;

  SetWRefCon(GetDialogWindow(dialogRef),(SInt32) kSheetDialog);

  SetDialogDefaultItem(dialogRef,kStdOkItemIndex);
```

```
  GetDialogItemAsControl(dialogRef,iEditTextSheetDialog,&controlRef);
  SetDialogItemText((Handle) controlRef,stringData);
  SelectDialogItemText(dialogRef,iEditTextSheetDialog,0,32767);

  gSheetDialogWindowRef = GetDialogWindow(dialogRef);
  osError = ShowSheetWindow(gSheetDialogWindowRef,gWindowRef);

  menuRef = GetMenuRef(mDemonstration);
  if(menuRef != NULL)
    DisableMenuItem(menuRef,iWindowModalDialog);

  return (osError == noErr);
}
// ************************************************************
doButtonHitInSheetDialog

void  doButtonHitInSheetDialog(void)
{
  DialogRef  dialogRef;
  ControlRef controlRef;
  GrafPtr    oldPort;

  dialogRef = GetDialogFromWindow(gSheetDialogWindowRef);

  GetDialogItemAsControl(dialogRef,iEditTextSheetDialog,&controlRef);
  GetDialogItemText((Handle) controlRef,gCurrentString);

  HideSheetWindow(gSheetDialogWindowRef);
  DisposeDialog(dialogRef);
  gSheetDialogWindowRef = NULL;

  GetPort(&oldPort);
  SetPortWindowPort(gWindowRef);
  doDrawMessage(gWindowRef,true);
  SetPort(oldPort);
}

//
****************************************************************************************
// HelpTags.c
//
****************************************************************************************

// ..........................................................................
includes

#include "DialogsAndAlerts.h"
#include <string.h>

// ************************************************************************
helpTagsModal

void  helpTagsModal(DialogRef dialogRef)
{
  HMHelpContentRec helpContent;
  SInt16           a;
  static SInt16    itemNumber[7] = {  1,2,3,7,8,10,11 };
  ControlRef       controlRef;

  memset(&helpContent,0,sizeof(helpContent));

  HMSetTagDelay(500);
  HMSetHelpTagsDisplayed(true);

  helpContent.version = kMacHelpVersion;
```

```
  helpContent.tagSide = kHMOutsideTopCenterAligned;
  helpContent.content[kHMMinimumContentIndex].contentType = kHMStringResContent;
  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmResID = 129;

  for(a = 1;a <= 7; a++)
  {
    helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = a;
    GetDialogItemAsControl(dialogRef,itemNumber[a - 1],&controlRef);
    HMSetControlHelpContent(controlRef,&helpContent);
  }
}

// ************************************************************************
helpTagsMovableModal

void  helpTagsMovableModal(DialogRef dialogRef)
{
  HMHelpContentRec helpContent;
  SInt16           a;
  static SInt16    itemNumber[9] = {  1,2,3,4,6,11,12,13,14 };
  ControlRef       controlRef;

  memset(&helpContent,0,sizeof(helpContent));

  HMSetTagDelay(500);
  HMSetHelpTagsDisplayed(true);

  helpContent.version = kMacHelpVersion;
  helpContent.tagSide = kHMOutsideTopCenterAligned;
  helpContent.content[kHMMinimumContentIndex].contentType = kHMStringResContent;
  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmResID = 130;

  for(a = 1;a <= 9; a++)
  {
    helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = a;
    GetDialogItemAsControl(dialogRef,itemNumber[a - 1],&controlRef);
    HMSetControlHelpContent(controlRef,&helpContent);
  }
}

// ************************************************************************
helpTagsModeless

void  helpTagsModeless(DialogRef dialogRef)
{
  HMHelpContentRec helpContent;
  SInt16           a;
  static SInt16    itemNumber[7] = {  1,2,3,4,5,6,7 };
  ControlRef       controlRef;

  memset(&helpContent,0,sizeof(helpContent));

  HMSetTagDelay(500);
  HMSetHelpTagsDisplayed(true);

  helpContent.version = kMacHelpVersion;
  helpContent.tagSide = kHMOutsideTopCenterAligned;
  helpContent.content[kHMMinimumContentIndex].contentType = kHMStringResContent;
  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmResID = 131;

  for(a = 1;a <= 7; a++)
  {
    helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = a;
    GetDialogItemAsControl(dialogRef,itemNumber[a - 1],&controlRef);
    HMSetControlHelpContent(controlRef,&helpContent);
  }
```

```
}

//
**********************************************************************************************
```

Demonstration Program DialogsAndAlerts Comments

When this program is run, the user should

- Invoke alerts and dialogs by choosing items in the Demonstration menu, noting window update/activation/deactivation and menu enabling/disabling effects.
- On Mac OS 8/9, choose Show Balloons from the Help menu and pass the cursor over the various items in the dialogs, noting the information in the help balloons. Also note the updating of alerts and dialogs, and of the window, behind the help balloon when the balloon closes.
- On Mac OS X, pause the cursor over the various items in the dialogs, noting the information in the help tags.
- Note the effects on the menus when the various alerts and dialogs are the front window.
- Click anywhere outside the modal alert and modal dialog when they are the frontmost window, noting that the only response is the system alert sound.
- Note that, when the movable modal alert and movable modal dialog are displayed
    - The program can be sent to the background by clicking outside the alert or dialog and the document window, or by bringing another application to the foreground.
    - The program can be brought to the foreground again by clicking inside the alert or dialog, or the document window.
- Note that, when the movable modal dialog and (on Mac OS X) the window-modal (sheet) dialog are displayed, the Edit menu and its Cut, Copy, and Paste items are enabled, given that the edit text items in these dialogs always have keyboard focus.
- Note that, when the modeless dialog is displayed
    - It behaves like a normal document window when the user:
        - Clicks outside it when it is the frontmost window.
        - Clicks inside it when it is not the frontmost window.

    - It can be hidden by clicking in the close box/button or by selecting Close from the File menu.

    - A modal alert, movable modal alert, modal dialog or movable modal dialog can be invoked "on top of" the modeless dialog.

    - The Edit menu and its Cut, Copy, Paste, and Clear items are enabled so as to support text editing in the edit text item.

- Note that all alerts and dialogs respond correctly to Return and Enter key presses, and that the modal alert, modal dialog and movable modal dialog also respond correctly to escape key and Command-period presses.

- Note that, when an alert or dialog is the frontmost window, the window and content region are deactivated, the latter evidenced by dimming of the text in the document window's window header.

- In the modal dialog, click on the checkboxes to change their settings, noting that the new settings are remembered when the dialog is dismissed using the OK button, but not remembered when the dialog is dismissed using the Cancel button. Also, choose items in the two pop-up menus, noting that the chosen item is displayed in the document window's window header.

- In the movable modal dialog, click on the radio buttons to change their settings, noting that the new setting is remembered when the dialog is dismissed using the OK button, but not remembered when the dialog is dismissed using the Cancel button. In the case of the clock control and edit text item, change the item/part with keyboard focus using the Tab key or by clicking in that item/part. In the case of the edit text item, enter text, and edit that text using the Edit menu's Cut, Copy, Paste, and Clear items and their Command-key equivalents. Note that the cursor shape changes whenever the cursor is moved over the edit text item.

- In the modeless dialog and (on Mac OS X) the window-modal (sheet) dialog, enter text, and edit that text using the Edit menu's Cut, Copy, Paste, and Clear items and their Command-key equivalents. Note that, because no cursor adjustment function is included in the program, the cursor shape does not change whenever the cursor is moved over the edit text item. Also note that, when the Search button is clicked (or the Return or Enter keys are pressed) the text in the edit text item is displayed in the document window's window header.

- On Mac OS X, when a window-modal (sheet) alert or dialog is showing, minimize the window into the Dock and then expand the window from the dock.

In the `'DITL'` resources for the modal and movable modal dialogs, note that the item numbers of the primary group box items are lower than the item numbers of the items

visually contained by those group box items. This is to ensure that the group boxes do not draw over, and thus erase, the image of these contained items.

In the `'dlgx'` resources, note that all feature flags are set except for the `kDialogFlagsHandlesMovableModal` flag in `'dlgx'` resources for the modal dialog and modeless dialog. Thus dialogs have a root control and embedding hierarchy.

Although, for demonstration purposes, this program creates modal alerts and dialogs, it is emphasized that Aqua Human Interface Guidelines require that, on Mac OS X, applications use modal alerts and dialogs only in exceptional purposes. On Mac OS X, the vast majority of alerts and dialogs should be application-modal or window-modal.

On Mac OS X, explanatory Help tags are available for the modal, movable modal, and modeless dialogs. The associated source code (in the source code file HelpTags.c) is not explained in these comments because the provision of Help tags is incidental to the demonstration. For information on creating help tags, see Chapter 25. Note that it is also possible to display help tags on Mac OS 8/9; however, it is considered by the author that their "look" is somewhat at odds with the Platinum appearance, and that help balloons remain the most appropriate option for Mac OS 8/9.

The CodeWarrior project for this program adds the `CarbonFrameWorksLib` stub library because certain functions are used that are only available on Mac OS X. (See "Carbon and Available APIs" in Chapter 25.) Those functions (`CreateStandardAlert`, `RunStandardAlert`, `GetStandardAlertDefaultParams`, `CreateStandardSheet`, `ShowSheetWindow`, `HideSheetWindow`) are only called when the program is run on Mac OS X.

DialogsAndAlerts.h

Defines

Constants are established for dialog resource IDs and for the item numbers of certain items in the item lists associated with the dialogs. `rAlertStrings` and `rSheetStrings` represent the resource IDs of `'STR#'` resources holding strings for the message and informative text (label and narrative text in Mac OS 8/9 parlance) for the modal alert, movable modal alert, and window-modal (sheet) alert.

The values represented by `kSearchModeless` and `kSheetDialog` will be stored as the reference constant in the window object in, respectively, the modeless dialog and window-modal (sheet) dialog objects. This enables the program to distinguish between these two dialogs.

The penultimate block establishes constants representing the character codes for the

Return, Enter, escape, and period keys.

Finally, `MAX_UINT32` is defined as the maximum possible unsigned long value. This value will be assigned to `WaitNextEvent`'s sleep parameter at program launch.

DialogsAndAlerts.c

Global Variables

`gEventFilterUPP` will be assigned a universal procedure pointer to an application-defined event filter (callback) function. `gSleepTime` will be assigned the value to be used as the sleep parameter in the `WaitNextEvent` call. (This value will be changed during program execution.)

The three variables after `gDone` will store the current control value of the checkboxes in the modal dialog. The next variable will store the item number of the currently selected radio button in the movable modal dialog.

Finally, the pointer to the dialog object for the modeless dialog is declared as a global variable.

main

`GetNewDialog` is called to create a small modal dialog. `SetDialogTimeout` is then called with 10 (seconds) passed in the `inSecondsToWait` parameter and 1 passed in the `inButtonToPress` parameter. (In the associated `'DITL'` resource, Item 1 is the OK push button, which has been hidden.) The use of `SetDialogTimeout` requires that the application handle events for the dialog through the `ModalDialog` function, hence the `ModalDialog` do-while loop. This allows the Dialog Manager to simulate an item selection. After 10 seconds, the Dialog Manager simulates a user click in the (invisible) OK button, causing the do-while loop to exit. The dialog is then disposed of. (Note that pressing the Return key before the 10 seconds has elapsed will also dispose of the dialog.)

The call to `NewModalFilterUPP` creates a universal procedure pointer for the event filter (callback) function.

Note that error handling here and in other areas of this demonstration program is somewhat rudimentary. In the unlikely event that certain calls fail, `ExitToShell` is called to terminate the program.

eventLoop

The variable that will be used as `WaitNextEvent`'s sleep parameter (`gSleepTime`) is initially set to the maximum unsigned long value. Note that the value assigned to `gSleepTime` will be changed at certain points in the program.

When a `NULL` event is returned by `WaitNextEvent`, if the program is running on Mac OS 8/9, doIdle is called.

doIdle

`doIdle` is invoked, on Mac OS 8/9 only, whenever `WaitNextEvent` returns a null event.

If the front window is the modeless dialog, the function `IdleControls` is called. `IdleControls` calls the control definition function of those controls in the specified window which do idle-time processing. In this case, the control is an edit text control, and the call causes the control definition function to call `TextEdit` to blink the insertion point caret. (This call is not necessary on Mac OS X because controls on Mac OS X have built-in timers.)

doEvents

`doEvents` switches according to the event type received. (It is important to remember at this point that events that occur when the modal dialog or movable modal dialog have been invoked are not handled by the main event loop but by the `ModalDialog` function.)

Note that, at the `autoKey` case, the function `doKeyDown` is called only if the Command key is not down. This is to prevent the Command-key equivalents for cut, copy, and paste from repeating when the user presses and holds down those Command-key equivalents while editing text in the modeless dialog's edit text item.

In this program, `autoKey` events generated while the Command key is down are also discarded in the event filter (callback) function `eventFilter` (see below). This means that the behavior of the edit text item in the movable modal dialog will replicate that of the edit text item in the modeless dialog.

(Many commercial and shareware programs (BBEdit excepted) do not discard `autoKey` events in these circumstances. The author considers that to be an oversight.)

doMouseDown

`doMouseDown` handles mouse-down events. Mouse-downs in the content region and in the close box/button are of significance to the demonstration. If a mouse-down occurred in a close box/button, if `TrackGoAway` returns true, and if the window is the modeless dialog, `doHideModelessDialog` is called. (In this demonstration, the modeless dialog, but not the document window, has a close box/button.)

doKeyDown

`doKeyDown` handles all key-down and auto-key events.

First, the character code is extracted from the message field of the event structure. Then `IsDialogEvent` is called to determine whether the event occurred in the modeless dialog or window-modal (sheet) dialog, or in the document window.

If the event occurred in a document window, and if the modifiers field of the event structure indicates that the Command key was down, the function for adjusting the menus is called, `MenuEvent` is called to return the long value containing the menu and menu item associated with the Command-key equivalent, and the long value is passed to `doMenuChoice` for further handling.

If, however, the event occurred in the modeless dialog dialog or window-modal (sheet) dialog

- If the key pressed was the Return or Enter key, `GetDialogItemAsControl` is called to get a reference to the single push button control in the modeless dialog (item 1 in the item list). The push button is then highlighted for eight ticks (this has an effect on Mac OS 8/9 only), and then unhighlighted before a function is called to extract the text from the edit text item and display it in the document window's window header. `doKeyDown` then returns because it is not intended that the edit text item receive Return and Enter key presses.
- If the Command key was down
    - If either the X, C, or V key was pressed (that is, the user has pressed the Cut, Copy, or Paste Command-key equivalent), DialogSelect is called to further handle the event. `DialogSelect` uses `TextEdit` to cut, copy, or paste the text in the edit text item. (The calls to `HiliteMenu` briefly highlight the Edit menu to indicate to the user that an Edit menu Command-key equivalent has just been used. This replicates the highlighting that `ModalDialog` performs when Command-key presses occur in modal and movable modal dialogs with edit text items.)
    - If neither the X, C, nor V key was pressed, the function for adjusting the menus is called, `MenuEvent` is called to return the long value containing the menu and menu item associated with the Command-key equivalent, and the long value is passed to `doMenuChoice` for further handling.
    - `doKeyDown` returns so as to bypass the second call to `DialogSelect`.

  Thus the Command-key equivalents other than those for Cut, Copy, and Paste remain available to the user via the main event loop, while the Command-key equivalents for Cut, Copy, and Paste are trapped and passed to `DialogSelect` for handling.

- If the Return key and the Enter key were not pressed, and if the Command key was

not down, `DialogSelect` is called to handle the keystroke in conjunction with `TextEdit`, the visual result being that the character appears in the edit text item.

doUpdate

`doUpdate` performs the initial handling of update events.

If the call to `IsDialogEvent` reveals that the event is for a window of the document kind, a function for updating the document window is called.

If the event is for the modeless dialog or window-modal (sheet) dialog, `DialogSelect` is called to handle the event. `DialogSelect` calls `BeginUpdate`, `DrawDialog`, and `EndUpdate` to redraw the dialog's content area. To restrict the redraw to the update region, an alternative is to call `BeginUpdate`, `UpdateDialog`, and `EndUpdate`. (Recall here that update regions `BeginUpdate` and `EndUpdate` are irrelevant on Mac OS X.)

Note that, on Mac OS X, the call to `DialogSelect` is really only necessary in the case of the window-modal (sheet) dialog.

doUpdateDocument

`doUpdateDocument` simply fills the content region of the document window with a color, using a pixel patter (`'ppat'`) resource and a call to `FillCRect` for that purpose, and then calls a function which draws a window header frame and the current contents of `gCurrentString`.

doActivate

`doActivate` performs initial handling of activate events. If the call to `IsDialogEvent` reveals that the event is for a window of the document kind, the function for activating/deactivating the document window is called, otherwise the function for activating/deactivating the modeless dialog is called.

doActivateDocument

`doActivateDocument` performs window activation/deactivation for the document window. If the window is becoming active, the menus are adjusted as appropriate for a document window. The call `doDrawMessage` draws a window header frame in the window, and the current contents of `gMessageString`, in either the activated or deactivated mode.

doActivateDialogs

`doActivateDialogs` performs window activation and deactivation for the modeless dialog and window-modal (sheet) dialog.

`DialogSelect` is called to handle the event. If the modeless or window-modal (sheet) dialog is becoming active, `DialogSelect` activates all controls and, on Mac OS 8/9, redraws the one-pixel-wide modeless dialog frame in the active mode. If the modeless dialog is going to the back, `DialogSelect` deactivates all controls and, on Mac OS 8/9, redraws the one-pixel-wide modeless dialog frame in the inactive mode.

In the remaining code, if either the modeless dialog or window-modal (sheet) dialog is becoming active, the menus are adjusted and the global variable used in the sleep parameter of the `WaitNextEvent` function is assigned the value returned by `GetCaretTime` (the cursor-blinking interval set by the user in the General Controls control panel (Mac OS 8/9) and System preferences (Mac OS X)). This is necessary to ensure that null events will always be generated, and thus doIdle and `IdleControls` will be called (necessary on Mac OS 8/9 only), at an interval short enough to ensure insertion point caret blinking at the proper rate.

If either the modeless dialog or window-modal (sheet) dialog is being deactivated, the sleep parameter for the `WaitNextEvent` function is reset to the maximum unsigned long value.

(Recall that changing the value in `gSleepTime` is irrelevant on Mac OS X because the function `doIdle` will not be called when the program is running on Mac OS X.)

doAdjustMenus

Note that, if the program is running on Mac OS X, and a call to `IsWindowCollapsed` reveals that the document window has been minimized to the dock, the menu items which invoke the window-modal (sheet) alert and (sheet) dialog are disabled.

doMenuChoice

`doMenuChoice` handles menu choices. If the choice was the Close item in the File menu, and if the front window is the modeless dialog, a function that hides that modeless dialog is called. (In this program, because the document window does not have a close box/button, the Close item is only enabled when the modeless dialog is the front window.)

doEditMenu

`doEditMenu` first determines whether the front window is the modeless dialog or window-modal (sheet) dialog (both of which have an edit text item). If so, a reference to the dialog is obtained, following which Dialog Manager functions are called to cut, copy,

paste, or clear text as appropriate. The Dialog Manager, in conjunction with `TextEdit`, performs these operations.

doDemonstrationMenu

`doDemonstrationMenu` handles choices from the Demonstration menu, switching according to the menu item passed to it. Error handling in this function is somewhat rudimentary in that the program simply terminates.

Note that, when the Modal Alert item is chosen, `doModalAlerts` is called with false is passed in the function's parameter. Note also that, when the Movable Modal Alert item is chosen, `doMovableModalAlertonX` is called if the program is running on Mac OS X, otherwise `doModalAlerts` is called with true passed in the function's parameter. This latter reflects the fact that, on Mac OS X, the movable modal alert will be created by a function that is not available on Mac OS 8/9.

doExplicitlyDeactivateDocument

`doExplicitlyDeactivateDocument` is called at the beginning of those functions that create modal and movable modal alerts and dialogs.

If there is at least one window of any type open, and if that front window is of the document kind, the function for activating/deactivating document windows is called to deactivate the window.

doModalAlerts

`doModalAlerts` creates, displays, manages, and disposes of the modal alert and, on Mac OS 8/9, the movable modal alert.

The call to `doExplicitlyDeactivateDocument` deactivates the document window.

At the next nine lines, values are assigned to the fields of a standard alert parameter structure. In sequence: the alert is to be modal or movable modal depending on the value received in the formal parameter movable; a help button is to be displayed; the event filter (callback) function used is to be the application-defined event filter (callback) function pointed to by the universal procedure pointer `gEventFilterUPP`; the default title for the OK push button is to be used; a Cancel push button is required, and is to have the default title for the Cancel button; an Other push button is required, and is to have the title "Other"; the default push button is to be the first push button (which will thus have the default ring drawn around it (Mac OS 8/9) or be pulsing blue (Mac OS X) and have the Return and Enter keys aliased to it); the Cancel push button is to be the second push button (which will thus have escape and Command-period key presses aliased to it); the alert is to be displayed in the alert position on the parent window screen. (With regard to the last

field, the constant `kWindowDefaultPosition` equates to `kWindowAlertPositionParentWindowScreen`.)

The calls to `GetIndString` retrieve the specified strings from the specified `'STR#'` resource. These are passed in the `inError` and `inExplanation` parameters in the following call to `StandardAlert`.

The call to `StandardAlert` creates and displays the alert (specifying a Stop alert in the first parameter), and handles all user interaction (by internally calling `ModalDialog`), including dismissing the alert when either the OK, Cancel, or Other button is hit. The item hit is returned in `StandardAlert`'s `outItemHit` parameter.

In a real application, the appropriate action would be taken, based on which push button was hit, following the call to `StandardAlert`; however, in this demonstration, the identity of the push button is simply drawn in the document window.

doMovableModalAlertOnX

`doMovableModalAlertOnX` creates, displays, manages, and disposes of the movable modal alert when the program is run on Mac OS X. (The functions used in `doMovableModalAlertOnX` to create, display, and manage the alert are not available on Mac OS 8/9.)

The call to `doExplicitlyDeactivateDocument` deactivates the document window.

The call to `GetStandardAlertDefaultParams` initialises a standard `CFString` alert parameter structure with default values. (The defaults are: not movable; no Help button; no Cancel button; no Other button; alert position on parent window screen.) The next four lines modify the defaults by specifying that the alert is to be movable modal, and is to have a Help, Cancel, and Other, push button.

The two calls to `GetIndString` retrieve the specified strings from the specified `'STR#'` resource, which are then converted to `CFStrings` before being passed in the error and explanation fields in the following call to `CreateStandardAlert`. `CreateStandardAlert` creates the alert.

The call to `RunStandardAlert` displays the alert and runs the alert using a `ModalDialog` loop. When a push button is clicked, its item number is returned in the outItemHit parameter

In a real application, the appropriate action would be taken, based on which push button was hit, following the call to `RunStandardAlert`; however, in this demonstration, the identity of the push button is simply drawn in the document window.

doModalDialog

`doModalDialog` creates, displays, manages, and disposes of the modal dialog.

The call to `doExplicitlyDeactivateDocument` deactivates the document window.

The call to `GetNewDialog` creates the modal dialog from the specified resource as the frontmost window.

The call to `SetDialogDefaultItem` tells the Dialog Manager which is the default push button item and aliases the Return and Enter keys to that item. The call to `SetDialogCancelItem` tells the Dialog Manager which is the Cancel push button item, and aliases the escape key and Command-period key presses to that item.

The next block gets handles to the three checkbox controls and sets the value of those controls to the current values contained in the global variables relating to each control.

The call to `NewMenu` creates a new empty menu for a font menu. A reference to this menu is passed in the call to `CreateStandardFontMenu`, which creates a non-hierarchical font menu containing the names of all resident fonts. `GetDialogItemAsControl` gets a reference to the Font pop-up menu button control, facilitating the calls to `SetControlMinimum`, `SetControlMaximum`, and `SetControlData`. The latter sets the menu to be used by the pop-up menu button control.

With the modal dialog fully prepared, it is made visible by the call to ShowWindow.

The do/while loop continues to execute until `ModalDialog` reports that either the OK or Cancel button has been "hit." Within the loop, `ModalDialog` retains control until one of the enabled items has been hit.

If a checkbox is hit, `GetDialogItemAsControl` is called to get a reference to the control and `SetControlValue` is called to flip that control's control value. (If it is 0, it is flipped to 1, and vice versa.) If one of the pop-up menu buttons is hit, `GetDialogItemAsControl` is called to get a reference to the control and `GetControlValue` is called to get the menu item number of the menu item chosen, following which an function is called to extract the menu item text and display it in the window header.

Note that the first parameter in the `ModalDialog` call is a universal procedure pointer to the application-defined event filter (callback) function.

When the do/while loop exits, and if the user hit the OK button, handles to each of the three checkboxes are retrieved for the purposes of retrieving the control's value and assigning it to the relevant global variable. (If the user hit the Cancel button, the global

variables retain the values they contained before the dialog was created and displayed.)

The dialog is then disposed of.

doMovableModalDialog

`doMovableModalDialog` creates, displays, manages, and disposes of the movable modal dialog.

The call to `doExplicitlyDeactivateDocument` deactivates the document window.

The call to `GetNewDialog` creates the movable modal dialog from the specified resource as the frontmost window.

The call to `SetDialogDefaultItem` tells the Dialog Manager which is the default push button item and aliases the Return and Enter keys to that item. The call to `SetDialogCancelItem` tells the Dialog Manager which is the Cancel push button item and aliases the escape key and Command-period key presses to that item. The call to `SetDialogTracksCursor` tells the Dialog Manager to track the cursor and change it to the I-Beam cursor shape whenever it is over an edit text item.

The first call to `GetDialogItemAsControl` gets a reference to the radio button control represented by the current value in the global variable `gBrushType`. The value of that control is then set to 1. The second call to `GetDialogItemAsControl` gets a reference to the clock control. The call to `SetKeyboardFocus` sets the keyboard focus to that item.

Before the session of user interaction begins, the current value in the global variable `gBrushType`, which stores the item number of the currently selected radio button, is copied to the local variable `oldBrushType`. (This may be required later.)

With the movable modal dialog fully prepared, it is made visible by the call to `ShowWindow`.

The do/while loop continues to execute until `ModalDialog` reports that either the OK or Cancel button has been hit. Within the loop, `ModalDialog` retains control until one of the enabled items is hit.

If a radio button is hit, a for loop sets the control value of all radio button controls to 0. A call to `GetDialogItemAsControl` then gets a reference to the radio button control that was hit. A call to `SetControlValue` then sets that control's value to 1, and the item number of this radio button is assigned to the global variable `gBrushType`.

Note that the first parameter in the `ModalDialog` call is a universal procedure pointer to

the application-defined event filter (callback) function. Note also that all user interaction relating to the clock control and edit text item is handled automatically by `ModalDialog`, including the movement of keyboard focus between the items.

When the do/while loop exits, and if the user hit the Cancel button, the value stored in the local variable `oldBrushType` is assigned to `gBrushType`, ensuring that any change to the currently selected radio button within the do/while loop is ignored. (In a real application, a long date/time value from the clock control, and the text from the edit text item would possibly be retrieved at this point if the user hit the OK push button.)

doCreateOrShowModelessDialog

In this program, the modeless dialog is only created once, that is, when the user first chooses Modeless Dialog from the Demonstration menu. Clicks in its close box/button, or choosing Close from the File menu while the modeless dialog is the frontmost window, will cause the dialog to be hidden, not disposed of.

Accordingly, the first line determines whether the modeless dialog is already open. If it is not: the call to `GetNewDialog` creates the modeless dialog; the call to `SetWRefCon` assigns the reference constant `kSearchModeless` to the dialog object's window object so as to differentiate this dialog from the window-modal (sheet) dialog; a call to `SetDialogDefaultItem` causes the push button to be drawn with the default ring (Mac OS 8/9) or in pulsing blue (Mac OS X); a call to `SetDialogItemText` assigns some initial text to the edit text item; a call to `SelectDialogItemText` selects the text in the edit text item. (Note that if the edit text item did not contain text this latter call would simply display the insertion point caret, which would be made to blink by the call to `IdleControls` within the function doIdle (Mac OS 8/9).)

If, on the other hand, the modeless dialog has already been opened, the call to `ShowWindow` displays the dialog and the call to `SelectWindow` generates the necessary activate events.

doInContent

`doInContent` continues the content region mouse-down handling initiated by `doMouseDown`. `doInContent` is called by `doMouseDown` only if the mouse-down occurred in the frontmost (active) window.

If the event occurred in the document window, the mouse-down event would be handled in the if section of the if/else block. (No action is required in this demonstration.)

If the event occurred in the modeless dialog or the window-modal (sheet) dialog (both of which contain an edit text item), `DialogSelect` is called to handle the event. `DialogSelect` tracks enabled controls (only the push button is enabled), returning true

if the mouse button is released while the cursor is still inside the control, and highlights any selection made in the edit text item. If `DialogSelect` returns true, and if the item hit was the OK push button, a function is called to perform the actions required in the event of a hit on that button.

doButtonHitInSearchModeless

`doButtonHitInSearchModeless` further processes, to completion, a hit on the OK (Search) button in the modeless dialog. It simply demonstrates retrieval of the text in an edit text item in a dialog.

The call to `GetDialogItemAsControl` gets a reference to the edit text control, and the call to `GetDialogItemText` copies the text in the edit text control to the global variable `gCurrentString`. The following lines cause that text to be drawn in the window header in the document window.

doHideModelessDialog

`doHideModelessDialog` hides a modeless dialog. The call to `HideWindow` makes the dialog invisible. The sleep parameter for the `WaitNextEvent` function is reset to the maximum possible long value (relevant only on Mac OS 8/9), because insertion point caret blinking is not required while the Search dialog is hidden.

eventFilter

`eventFilter` is the application-defined event filter (callback) function which, in conjunction with `ModalDialog`, handles events in the modal alerts, movable modal alert on OS 8/9, modal dialog and movable modal dialog.

If the program is running on Mac OS 8/9, if the event is an update event, and if that event is not for the dialog or alert in question, the application's document window updating function is called and false is returned. This response to an update event in the application's own document windows also allows `ModalDialog` to perform a minor switch when necessary so that background applications can update their windows as well. The call to the window updating function is not necessary on Mac OS X.

If the event is an `autoKey` event and the Command key is down, the event is, in effect, discarded. This means that, if the user is working within the edit text item in the movable modal dialog and presses and holds the Command key equivalents for Cut, Copy or Paste, repeating cut, copy and paste actions will be defeated. That is, pressing and holding the Command key equivalent will only result in a single cut, copy, or paste action. (See also `doEvents`, above.)

If the event is neither an update event nor an `autoKey` event with the Command key

down, the current graphics port is saved and then set to that of the alert or dialog. The event is then passed to the standard event filter (callback) function for handling. If the standard event filter (callback) function handles the event, it will return true and, in the itemHit parameter, the number of the item that it handled. `ModalDialog` will then return this item number. A call to `SetPort` then restores the previously save graphics port.

Note that the calls to `GetPort` and `SetPort` are actually redundant when this event filter (callback) function is used by all but the movable modal dialog. The calls are only necessary when `SetDialogTracksCursor` has been called to cause the Dialog Manager to automatically track the cursor, and the movable modal dialog is the only dialog which requires this tracking (because it contains an edit text item.)

doPopupMenuChoice, doPlaySound, doDrawMessage, and doCopyPString

`doPopupMenuChoice`, `doPlaySound`, `doDrawMessage`, and `doCopyPString` are incidental to the demonstration. All perform the same duties as the similarly-named functions in the demonstration program Controls1 (Chapter 7). `doDrawMessage` is used in this program to prove the explicit deactivation of the document window's content area when alerts and dialogs other than the modeless dialog are invoked.

Sheets.c

Global Variables

`gSheetDialogWindowRe` will be assigned the window reference of the window-modal (sheet) dialog.

doSheetAlert

`doSheetAlert` creates, displays, and handles a window-modal (sheet) alert.

The call to `GetStandardAlertDefaultParams` initializes a standard `CFString` alert parameter structure with default values.

The two calls to `GetIndString` retrieve the specified strings from the specified `'STR#'` resource, which are then converted to `CFStrings` before being passed in the error and explanation fields in the following call to `CreateStandardSheet`, which creates the alert. The call to `ShowSheetWindow` displays the sheet. When the user clicks the OK button, the sheet is dismissed.

doSheetDialog

`doSheetAlert` creates and displays a window-modal (sheet) dialog.

The call to `GetNewDialog` creates the dialog from the specified resource. The call to `SetWRefCon` assigns a value as the dialog window's reference constant. This is used elsewhere to differentiate the window-modal (sheet) dialog from the modeless dialog.

The call to `SetDialogDefaultItem` causes the push button to be drawn with the default ring (Mac OS 8/9) or in pulsing blue (Mac OS X). The call to `SetDialogItemText` assigns some initial text to the edit text item and the call to `SelectDialogItemText` selects that text.

The next line assigns a reference to the dialog's window to the global variable `gSheetDialogWindowRef`. This is used in the function `doButtonHitInSheetDialog`.

The call to `ShowSheetWindow` displays the sheet.

doButtonHitInSheetDialog

`doButtonHitInSearchModeless` is called from `doKeyDown` and `doInContent` to further process, to completion, a hit on the OK button in the window-modal (sheet) dialog. In addition to retrieving the text in an edit text item, it hides and disposes of the dialog.

Chapter 9. The Finder and the Application

IN THIS CHAPTER

- Introduction
- An Application's Required Resources
- Application Signature, File Creator, and File Types
- Icons for the Finder
- File Reference ('FREF') Resources
- The Bundle ('BNDL') Resource
- The Version ('vers') Resource
- Finder Icon Help ('hfdr') Resource-Mac OS 8/9
- Missing Application and Application-Missing Resources-Mac OS 8/9
- The 'plst' Resource and Information Property Lists
- Application Launching by the Finder
- The Finder and the Catalog File
- Preferences, Temporary Items, and Trash Folders-Using FindFolder
- Stationery Pads
- Using Aliases
- Mac OS 9 Packages and Mac OS X Bundles
- Relevant Constants, Data Types, and Functions
- Demonstration Programs

Introduction

The Finder

The **Finder** is an application that works with the system software to mediate the user's access to volumes, folders and files in the file system, present a visual representation of those volumes, folders, and files to the user on the desktop, and generally manage the desktop. The term desktop in this context simply means the work area on the screen.

The Finder requires that your application supply it with certain information. The Finder accesses this information and builds its own database of the resources it needs, such as the icons to use to display your application and the documents it creates.

The Finder uses certain high-level events known as **Apple events** to communicate certain instructions and information to your application. Accordingly, this chapter should be read in conjunction with Chapter 10.

Relevant Databases

The Desktop Database-Mac OS 8/9

When a file is created or installed, the File Manager extracts some of the information provided by resources described in this chapter and stores it in the volume's **catalog file,** a special file containing information about the hierarchical organization of files and folders on that volume.

Although it is mostly used by the File Manager, the information in the catalog file is also used by the Mac OS 8/9 Finder. The Finder extracts the information provided by your application and uses it to build a **desktop database.** The purpose of the desktop database is to facilitate rapid access by the Finder to icons, file types, applications, version data, and comments. The Finder builds the desktop databases for each mounted volume at system start-up, and updates them as files and directories are added, moved, renamed or deleted.

The desktop database:

- Contains the definitions of all icons and their associated file types.
- Lists all the file types that an application can open.
- Lists the location of each application on the disk.
- Contains any comments that the user has added to the information windows (the window opened when the user selects an icon and chooses Get Info from the Mac OS 8/9 Finder's File menu) for desktop objects.

The Finder's Private Databases-Mac OS X

Mac OS X maintains a number of private databases, one of which is the **application database.** Because of the multi-user nature of Mac OS X, the Finder maintains an application database for each user with an account on the system. This database contains information about all the applications the Mac OS X Finder has encountered for that user and includes information about the document types understood by each application.

If your application includes a `'plst'` (property list) resource, which itself includes a binary representation of an **information property list** containing the necessary information, the Mac OS X Finder extracts the information it requires from that list. If the `'plst'` resource is empty, the Mac OS X Finder extracts the information provided by the signature, file reference, bundle, and version resources described in this chapter.

> **Note**
>
> As will be seen, an application's `'plst'` resource must contain an information property list in order for it to specify large (128-by-128 pixel) thumbnail icons for itself and its documents. Since this will ordinarily be the requirement, the following assumes that the application includes an information property list containing all relevant information in its `'plst'` resource.
>
> Note also that, in so-called "bundled" Mac OS X applications, information property lists are provided in files, not in `'plst'` resources. However, the Mac OS X application packaging scheme known as the bundle is not addressed in this book.

The Mac OS 8/9 Finder ignores `'plst'` resources.

An Application's Required Resources

The Mac OS 8/9 Finder requires that your application provide the following:

- A **signature resource,** which enables the Finder to identify and start up your application when a user double clicks a document created by your application.
- **Icon resources,** which represent your application and any documents it creates to the user.
- **File reference resources,** which link icons with the file types they represent.
- A **bundle resource,** which groups signature, icon, and file reference resources together.
- A `'SIZE'` **resource** (see Chapter 2).

Other resources that should be provided by your application for Mac OS 8/9 are:

- **Version resources,** which allow users to ascertain the version of your application.
- A **help resource,** which is used by the Finder to display your application's customized balloon help message.

Assuming that the application's `'plst'` resource contains the necessary information, Mac OS X does not utilize the signature, file reference, bundle, or version resources. The help resource is irrelevant on Mac OS X.

Application Signature, File Creator, and File Types

Application Signature

Your application's **signature** is a unique four-character sequence, which must not conflict with that of any other application, and which the Finder uses to identify your application.

> **Note**
>
> To ensure uniqueness, developers are required to register their application's signature with Apple Computer, Inc, at Macintosh Developer Technical Support.

To provide its signature, your application must include a special resource whose resource type is the application's signature and whose resource ID is 0.

Creating an Application Signature Resource Using Resorcerer

Resorcerer allows you to create an application signature resource within its bundle resource editing window (see Fig 5). Fig 1 shows an application signature resource being created using Resorcerer.

Figure 5. STRUCTURE OF A COMPILED 'FREF' RESOURCE AND CREATING A 'FREF' RESOURCE USING RESORCERER

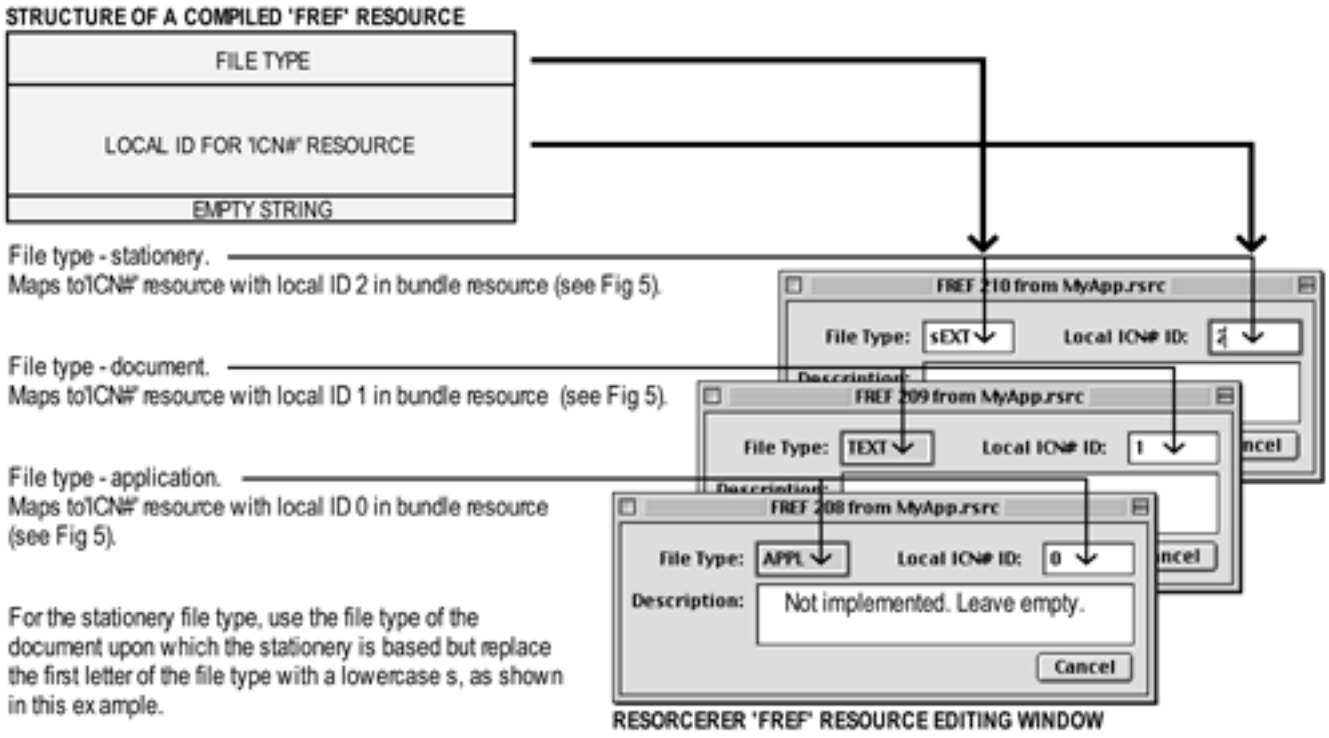
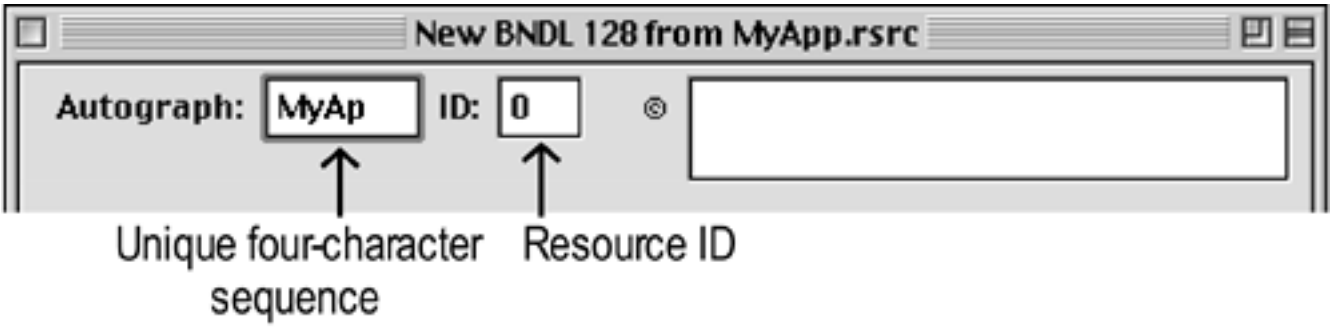

STRUCTURE OF A COMPILED 'FREF' RESOURCE
FILE TYPE
LOCAL ID FOR 'ICN#' RESOURCE
EMPTY STRING
File type - stationery.
Maps to 'ICN#' resource with local ID 2 in bundle resource (see Fig 5).
File type - document.
Maps to 'ICN#' resource with local ID 1 in bundle resource (see Fig 5).
File type - application.
Maps to 'ICN#' resource with local ID 0 in bundle resource (see Fig 5).
For the stationery file type, use the file type of the document upon which the stationery is based but replace the first letter of the file type with a lowercase s, as shown in this example.
FREF 210 from MyApp.rsrc
File Type: sEXT
Local ICN# ID:
FREF 209 from MyApp.rsrc
File Type: TEXT
Local ICN# ID: 1
FREF 208 from MyApp.rsrc
File Type: APPL
Local ICN# ID: 0
Description:
Not implemented. Leave empty.
Cancel
RESORCERER 'FREF' RESOURCE EDITING WINDOW

Figure 1. CREATING A SIGNATURE RESOURCE USING RESORCERER



File Creator and File Type

When an application creates a document, it assigns that document a **creator** and a **file type.** Your application should set its own signature as the creator of the documents it creates. The creator field of the application file itself should contain the application's own signature.

Use of the Creator by the Finder

The Finder reads the creator field of a document file when the user double clicks on the file's icon or selects it and chooses Open from the Finder's File menu (or, on Mac OS 8/9 only Print from the Finder's File menu). Having obtained the creator, the Finder then searches for an application with a signature of that name and, if it finds the application, calls the Process Manager to start it up. The Finder then passes to your application the information it needs to open or print the document via an Apple event.

Note

Note that in this and other references to your application receiving requests to open and print documents, the assumption is made that your application supports the receipt and handling of required Apple events (see Chapter 10).

File Type

The file type can be a type especially defined for your application or it can be one of the existing general types, some of which are as follows:

File Type	*Description*
`'APPL'`	Launchable application.
`'DRVR'`	Driver.
`'FFIL'`	File for storing fonts.
`'PICT'`	QuickDraw picture.
`'PRER'`	Printer driver.
`'TEXT'`	Stream of ASCII characters.
`'appe'`	Background only application.
`'ffil'`	Font.
`'pref'`	Preferences file.
`'sfil'`	Sound.
`'ttro'`	SimpleText read-only file.

Your application's file type must be `'APPL'`.

Note

Apple reserves the use of all signatures and file types whose names contain only lowercase and non-alphabetic characters. Your signature and file type created especially for your application must each contain at least one uppercase character. Like signatures, file types must be registered with Apple.

Use of the File Type by the Finder

Another way for a user to open a document created by your application, as well as a document not created by your application but which is of a file type *supported* by your application, is to select its icon and drag the selected icon to your application's icon. If the document's file type is supported by your application, the Finder calls the Process Manager to launch your application and then sends your application a required Apple event containing the information it needs to open the document.

Use of File Type by Your Application

Your application also relies on file types to determine which files to allow the user to open when your application is running. When your application calls Navigation Services to open a file, your application supplies either a list of the file types that your application can open or a filter function for those types. The Open dialog then displays files of the specified types only. (See Chapter 18.)

Icons for the Finder

If you do not supply your own icons, the Finder uses its own default application and document icons for display. Fig 2 shows some of the Finder's default icons.

Figure 2. DEFAULT ICONS



Although default icons are available, you should design and create your own icons for all the file types associated with your application, such as

- The application file.
- Documents created by your application.
- Stationery pads created by users from your application's documents. (Stationery pads are files the user creates to serve as templates for other documents.)

The Icon Family

An **icon family** is a collection of icons intended to represent a single file type. The provision of a family of icon types, rather than just one icon type, enables the Finder to automatically select the appropriate family member to display depending on the icon size specified by the user and the bit depth of the display device.

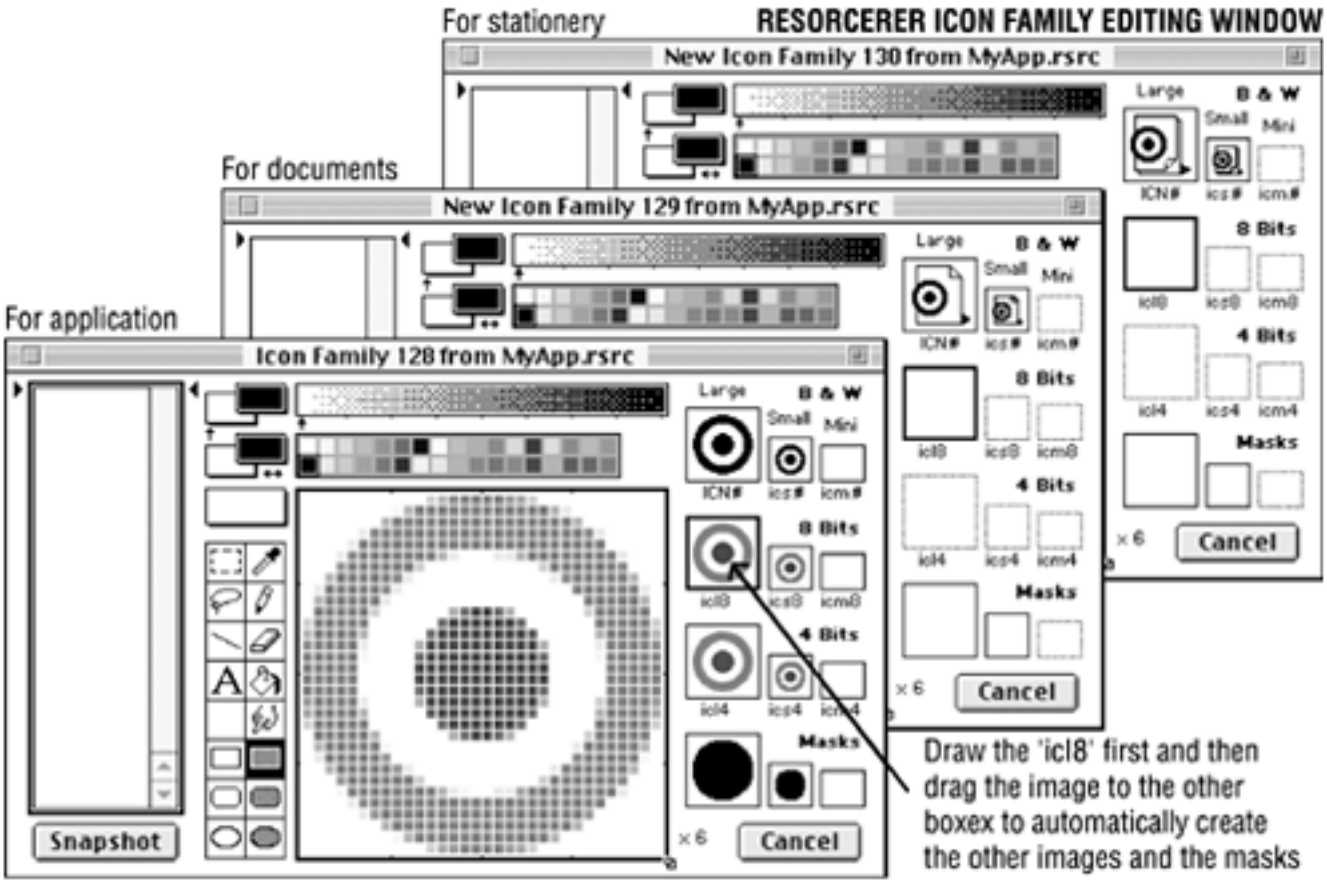
Prior to Mac OS 8.5, the following icons comprised the icon family for a single file:

Icon	***Size (Pixels)***	***Resource in Which Defined***
Large black-and-white icon (1-bit) and mask	32-by-32	Icon list (`'ICN#'`).
Small black-and-white icon (1-bit) and mask	16-by-16	Small icon list (`'ics#'`)
Mini black-and-white icon (1-bit and mask)	12-by-16	Mini icon list (`'icm#'`)

Large color icon with 4 bits of color data per pixel	32-by-32	Large 4-bit color icon (`'icl4'`)
Small color icon with 4 bits of color data per pixel	16-by-16	Small 4-bit color icon (`'ics4'`)
Mini color icon with 4 bits of color data per pixel	12-by-16	Mini 4-bit color icon (`'icm4'`)
Large color icon with 8 bits of color data per pixel	32-by-32	Large 8-bit color icon (`'icl8'`)
Small color icon with 8 bits of color data per pixel	16-by-16	Small 8-bit color icon (`'ics8'`)
Mini color icon with 8 bits of color data per pixel	12-by-16	Mini 8-bit color icon (`'icm8'`)

All of these icons use 1-bit masks, which are stored with the `'ICN#'`, `'ics#'`, and `'icm#'` resources. The Finder uses the mask to crop the icon's outline into the background and then draws the icon into this shape. For this reason, the mask must be exactly the same shape as the icon (see Fig 3).

Figure 3. THREE ICON FAMILIES BEING CREATED USING RESORCERER



Note that the term "icon list" applying to the names of the black-and-white icon resources is somewhat of a misnomer in that you can define only two images in these resources: the icon and its mask.

The mini icon is of doubtful utility in the Carbon era and will not be considered further.

Huge Icons, 32-Bit Icons, and Deep Masks

Mac OS 8.5 introduced the following:

- Huge (48-by-48 pixels) icons (`'ich#'`, `'ich4'`, `'ich8'`, and `'ih32'`).
- 32-bit small, large, and huge icons (`'is32'`, `'il32'`, and `'ih32'`).
- 8-bit masks for the 32-bit icons (`'s8mk'`, `'l8mk'`, and `'h8mk'`). These "deep" masks support 256 different levels of transparency.

The bundle resource, however, does not support huge icons, 32-bit icons, or 8-bit masks. On Mac OS 8/9, therefore, you cannot specify these icons for your application or its documents. You can, however include them in an `'icns'` resource (see below) for Mac OS X.

Thumbnail Icons, the 'icns' Resource, and Mac OS X

For Mac OS X, you should provide 128-by-128 pixel, 32-bit, **thumbnail** icons (`'it32'`) icons. The associated mask (`'t8mk'`) depth is 8-bits.

This icon should be included, along with other required icons, in an `'icns'` resource. (The `'icns'` resource was introduced with Mac OS 8.5 as a means of providing a single source for all icon data. Combining all icon data into a single resource type speeds up icon fetching and simplifies resource management.) The ID of the `'icns'` resource must be included in the information property list in your application's `'plst'` resource.

Mac OS X can generate icons of various sizes by scaling the thumbnail. Nevertheless, if you need to preserve fine details at smaller resolutions, you may provide large and small icons, in addition to thumbnail icons, in your `'icns'` resources. In this case, the system will use the best available icon for the display size required.

Creating Icon Resources

Unless you are using Resorcerer Version 2.4 or later, it will not be possible to use that application to create huge icons, any 32-bit icons, the thumbnail icons required for Mac OS X, or `'icns'` resources. In these circumstances, a reasonable methodology is to

- Use Resorcerer to create the icons that will be used when the application runs on Mac OS 8/9.

- Use the shareware program Iconographer to create the icons and `'icns'` resource that will be used when the application runs on Mac OS X.

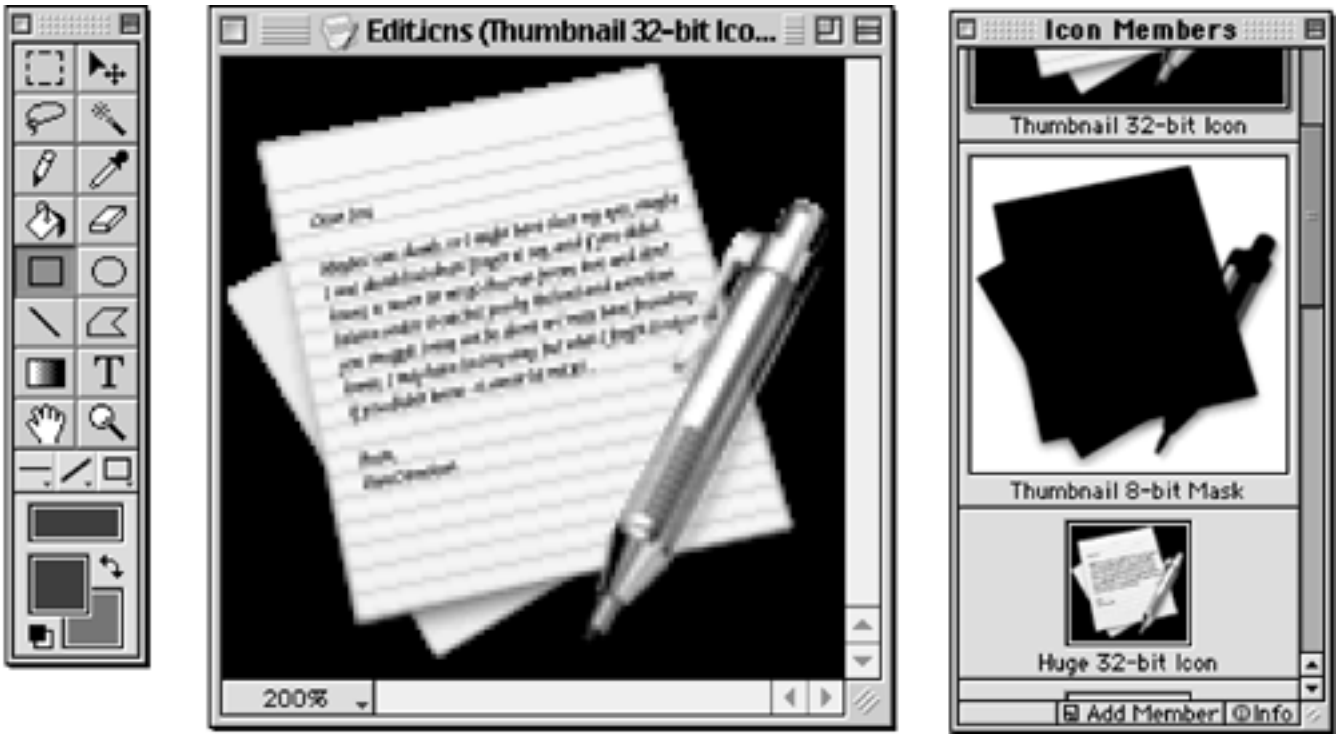
Creation Using Resorcerer

Fig 3 shows large and small 1-bit, 4-bit, and 8-bit icons being created in Resorcerer.

Creation Using Iconographer

Fig 4 shows an application's icons being created using Iconographer. You can copy the `'icns'` resource in the resource fork of the file created by Iconographer to your application's resource file.

Figure 4. AN APPLICATIONS ICON BEING CREATED USING ICONOGRAPHER



Icon Design-Small and Large Icons

For the small and large icons used by Mac OS 8/9, good icon design requires that one graphic element be repeated in all icons created for the application. This allows the user to quickly identify the files associated with your product.

Icon Design-Thumbnail Icons

The design philosophy for thumbnail icons is significantly different from that applying to icons for Mac OS 8/9. The large size, bit-depth, and transparency characteristics lend themselves to

the creation of icons with photographic realism and perspective effects, making the icon appear as if it is an object on a desk as viewed from a chair.

An icon for an application should contain a base object (such as, a sheet of water color paper containing a half-finished painting) and a supportive "tool" (such as a paint brush), which communicates the type of task that the application allows the user to accomplish. The effect should be such that the icon is perceived by the user as a familiar object viewed in a familiar way (that is, with perspective).

File Reference ('FREF') Resources

File reference (`'FREF'`) resources associate your application's icons with file types created and supported by your application, allowing users to open documents supported by your application by dragging their icons to your application's icon.

You create separate `'FREF'` resource for your application file and for each file type that your application can open.

Structure of the 'FREF' Resource and Creating a 'FREF' Resource Using Resorcerer

Fig 5 shows the structure of a `'FREF'` resource and a `'FREF'` resource being created using Resorcerer. The following describes the main fields of a compiled `'FREF'` resource:

Field	*Description*
FILE TYPE	A four-character code identifying the type of file represented by this resource.
LOCAL ID FOR 'ICN#' RESOURCE	Used by the Finder to map the file type specified in this resource to an icon list (`'ICN#'`) resource with the same local ID in the bundle (`'BNDL'`) resource.
EMPTY STRING	Not implemented.

Use of 'FREF' Resources by the Finder

The Finder maintains a list of your `'FREF'` resources. When, for example, the user drags a document icon to your application icon, the Finder checks the list and, if the document's file type appears in this list, launches your application with a request, passed via a required Apple event, to open that document.

You can define `'FREF'` resources for file types your application supports but for which it does not provide icons. This allows users to launch your application by dragging the icons representing documents of this type to your application icon. For example, you could create a `'FREF'` resource for the document file type `'ttro'`, which is used by documents that have SimpleText as their creator. (The Finder uses SimpleText's icon family resources for these documents.) The user can then open these documents in your application by dragging the document icon to your application's icon.

The Bundle ('BNDL') Resource

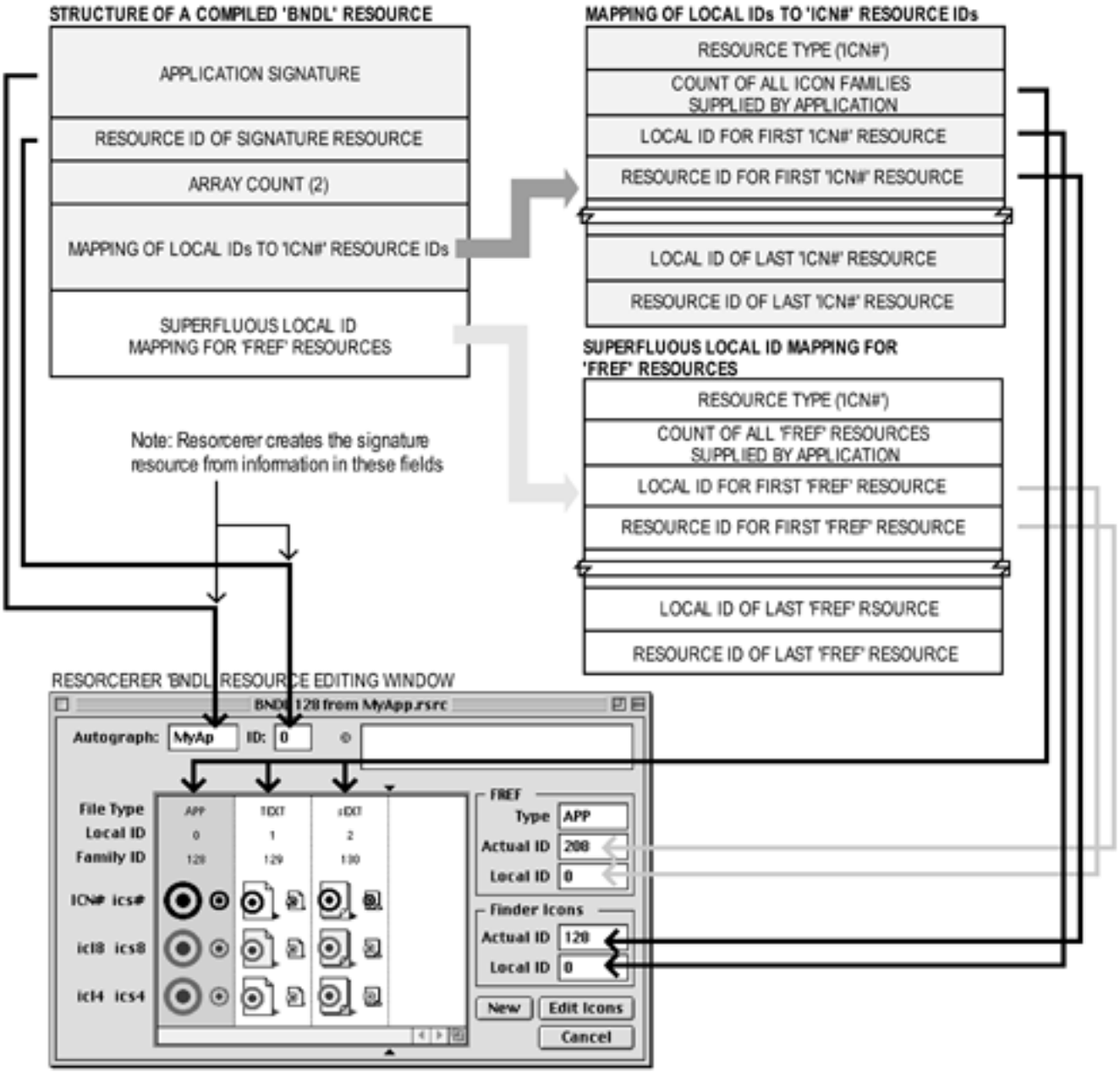
A bundle (`'BNDL'`) resource associates all of the resources (signature, icons, and file reference) used by the Finder for your application.

When the Finder displays your application on the desktop for the first time, it determines whether the application has a `'BNDL''` resource by checking the catalog file. If the Finder finds a 'BNDL' resource, it installs the information from the `'BNDL'` resource and its associated resources into the Mac OS 8/9 desktop database and uses that information to display icons for your application's file types. If the Finder does not find a `'BNDL'` resource, it displays your application using the default icons.

Structure of the 'BNDL' Resource and Creating a 'BNDL' Resource Using Resorcerer

Fig 6 shows the structure of a `'BNDL'` resource and a `'BNDL'` resource being created using Resorcerer.

Figure 6. STRUCTURE OF A COMPILED 'BNDL' RESOURCE AND CREATING A 'BNDL' RESOURCE USING RESORCERER


STRUCTURE OF A COMPILED 'BNDL' RESOURCE
APPLICATION SIGNATURE
RESOURCE ID OF SIGNATURE RESOURCE
ARRAY COUNT (2)
MAPPING OF LOCAL IDs TO 'ICN#' RESOURCE IDs
SUPERFLUOUS LOCAL ID MAPPING FOR 'FREF' RESOURCES
Note: Resorcerer creates the signature resource from information in these fields
MAPPING OF LOCAL IDs TO 'ICN#' RESOURCE IDs
RESOURCE TYPE ('ICN#')
COUNT OF ALL ICON FAMILIES SUPPLIED BY APPLICATION
LOCAL ID FOR FIRST 'ICN#' RESOURCE
RESOURCE ID FOR FIRST 'ICN#' RESOURCE
LOCAL ID OF LAST 'ICN#' RESOURCE
RESOURCE ID OF LAST 'ICN#' RESOURCE
SUPERFLUOUS LOCAL ID MAPPING FOR 'FREF' RESOURCES
RESOURCE TYPE ('ICN#')
COUNT OF ALL 'FREF' RESOURCES SUPPLIED BY APPLICATION
LOCAL ID FOR FIRST 'FREF' RESOURCE
RESOURCE ID FOR FIRST 'FREF' RESOURCE
LOCAL ID OF LAST 'FREF' RSOURCE
RESOURCE ID OF LAST 'FREF' RESOURCE
RESORCERER 'BNDL' RESOURCE EDITING WINDOW
Autograph: MyAp
ID: 0
File Type
Local ID
Family ID
ICN# ics#
icl8 ics8
icl4 ics4
FREF
Type APP
Actual ID 208
Local ID 0
Finder Icons
Actual ID 128
Local ID 0
New
Edit Icons
Cancel

The following describes the main fields of a compiled `'BNDL'` resource:

Field	***Description***
APPLICATION SIGNATURE	Identifies the application to the Finder.
RESOURCE ID OF THE SIGNATURE RESOURCE	Signature resource ID. (Always `0`.)
LOCAL ID OF FIRST 'ICN#' RESOURCE	Must match the local ID assigned to the `'ICN#'` resource within a `'FREF'` resource.

RESOURCE ID OF FIRST 'ICN#' RESOURCE	To visually represent files of the type described in the 'FREF' resource that contains the local ID in the preceding field, the Finder uses the appropriate member of the icon family with the same resource ID as this 'ICN#' resource.
LOCAL ID OF FIRST 'FREF' RESOURCE	A local ID for the 'FREF' resource relating to this file type. (Superfluous.)
RESOURCE ID OF FIRST 'FREF' RESOURCE	The resource ID of the 'FREF' resource relating to this file type. (Superfluous.)

Note that you also assign local IDs to 'FREF' resources within the 'BNDL' resource. This assignment is superfluous because the Finder does not map these local IDs to any other resources. It was implemented for the earliest versions of the system software, and it remains this way today for backward compatibility.

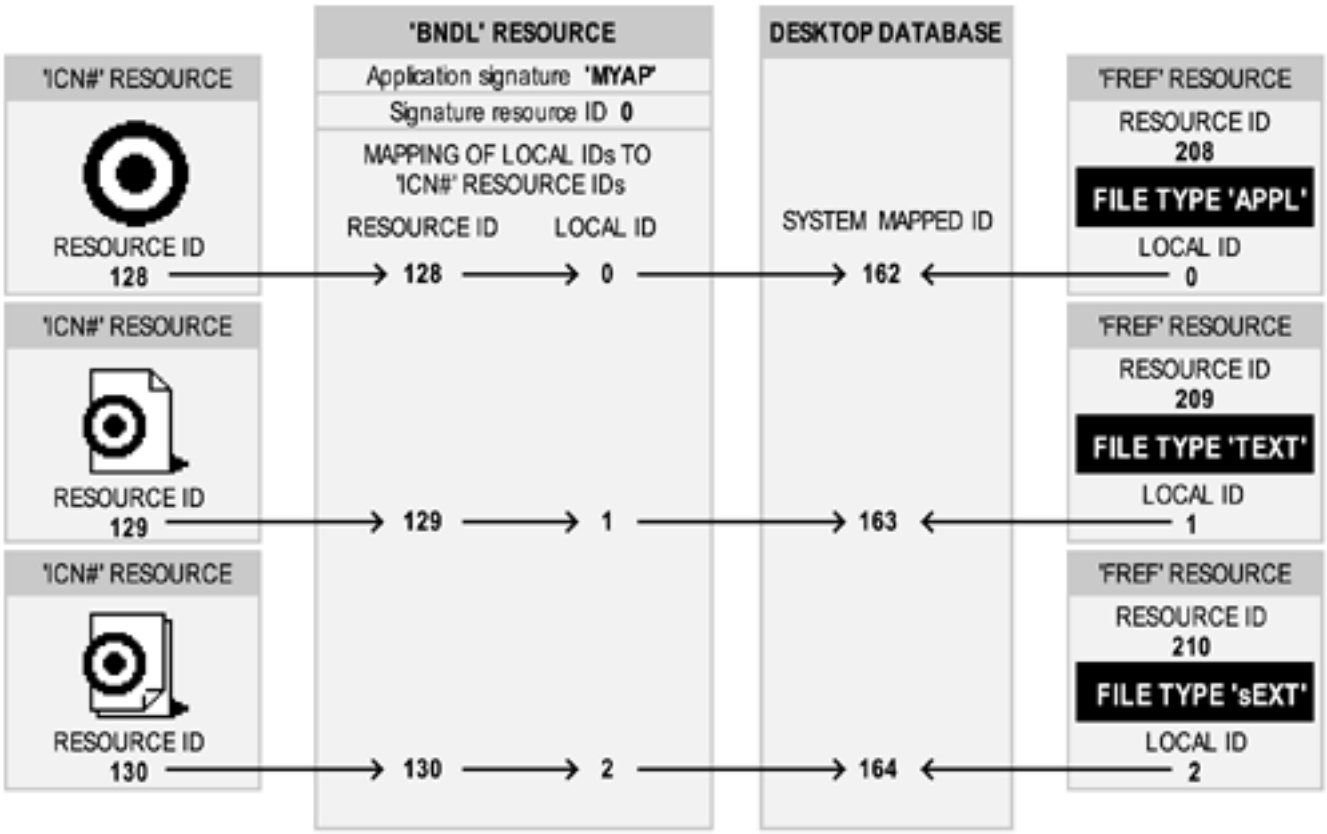
Resource IDs, Local Ids, and the Desktop Database

As shown in Fig 6, you must assign local IDs to your 'ICN#' resources within your 'BNDL' resources. You must ensure that, for all your file types for which you provide icons, these local IDs match the local IDs you assigned inside their corresponding 'FREF' resources.

In the desktop database, the Finder renumbers the resource IDs that you have assigned to your resources to avoid conflicts with the resources of other applications. Therefore, the 'BNDL' resource has to rely on these local IDs to map 'ICN#' resources to their 'FREF' resources, that is, the 'BNDL' resource uses the local ID you assign to an 'ICN#' resource to map it to the 'FREF' resource that has specified the same local ID. For example, the 'FREF' resource with resource ID 208 in Fig 5 shows that the file type 'APPL' is assigned a local ID of 0. In Fig 6, you will see that local ID 0 is assigned to the 'ICN#' resource with resource ID 128. This maps the icon defined by this resource to the application file.

Fig 7 illustrates how the 'BNDL' resource uses local IDs to map 'ICN#' resources to 'FREF' resources.

Figure 7. HOW THE BUNDLE RESOURCE LINKS 'ICN#' RESOURCES AND 'FREF' RESOURCES



In Fig 7, the application file's `'ICN#'` resource has a resource ID of 128 while its `'FREF'` resource has a resource ID of 208. In an actual application, it is best if you assign the same resource ID to a file's `'FREF'` resource that you assign to its `'ICN#'` resource. This makes code maintenance easier.

Informing the Finder that Your Application has a 'BNDL' Resource

You alert the Finder that your application has a `'BNDL'` resource by setting a bit in the file's Finder flags field (see below).

The Version ('vers') Resource

You use version (`'vers'`) resources to supply version information to the Finder. There are two ways that the Finder displays this information:

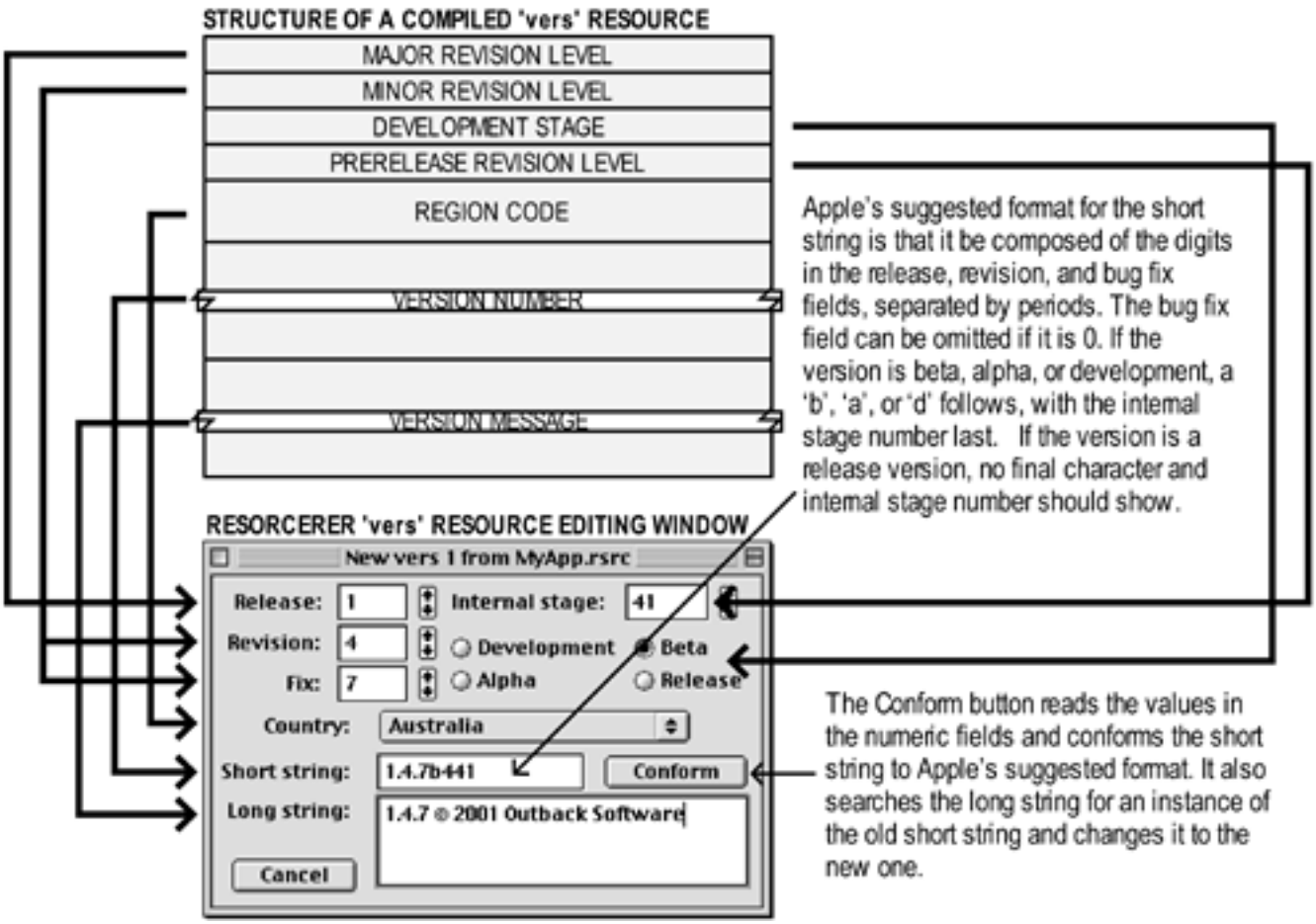
- In Finder list views, the version number (short string) is displayed in the Version column.
- The version message (long string) is displayed in the Finder's Get Info window on Mac OS 8/9 and in the Finder's Show Info window on Mac OS X.

You can use `'vers'` resources to assign version information to an individual file and, if it is a part of a larger collection of files, to the entire superset of files. The `'vers'` resource with ID 1 specifies the version of the file; the `'vers'` resource with ID 2 specifies the version of the set of files.

Structure of the 'vers' Resource and Creating a 'vers' Resource Using Resorcerer

Fig 8 shows the structure of a `'vers'` resource and a `'vers'` resource being created using Resorcerer.

Figure 8. STRUCTURE OF A COMPILED 'vers' RESOURCE AND CREATING A 'vers" RESOURCE USING RESORCERER


STRUCTURE OF A COMPILED 'vers' RESOURCE
MAJOR REVISION LEVEL
MINOR REVISION LEVEL
DEVELOPMENT STAGE
PRERELEASE REVISION LEVEL
REGION CODE
VERSION NUMBER
VERSION MESSAGE
Apple's suggested format for the short string is that it be composed of the digits in the release, revision, and bug fix fields, separated by periods. The bug fix field can be omitted if it is 0. If the version is beta, alpha, or development, a 'b', 'a', or 'd' follows, with the internal stage number last. If the version is a release version, no final character and internal stage number should show.
RESORCERER 'vers' RESOURCE EDITING WINDOW
New vers 1 from MyApp.rsrc
Release: 1
Internal stage: 41
Revision: 4
Development
Beta
Fix: 7
Alpha
Release
Country: Australia
Short string: 1.4.7b441
Conform
Long string: 1.4.7 © 2001 Outback Software
Cancel
The Conform button reads the values in the numeric fields and conforms the short string to Apple's suggested format. It also searches the long string for an instance of the old short string and changes it to the new one.

The following describes the main fields of a compiled `'vers'` resource:

Field	***Description***
MAJOR REVISION LEVEL	Major revision level in binary coded decimal format. Although the Finder does not display it anywhere, you can store this information here.
MINOR REVISION LEVEL	Major revision level in binary coded decimal format. Although the Finder does not display it anywhere, you can store this information here.
DEVELOPMENT STAGE	The values in this field indicate one of four development stages: development, alpha, beta, or released.
PRERELEASE REVISION LEVEL	The version if the software is still in prerelease.
REGION CODE	The script system for which this version of the software is intended.

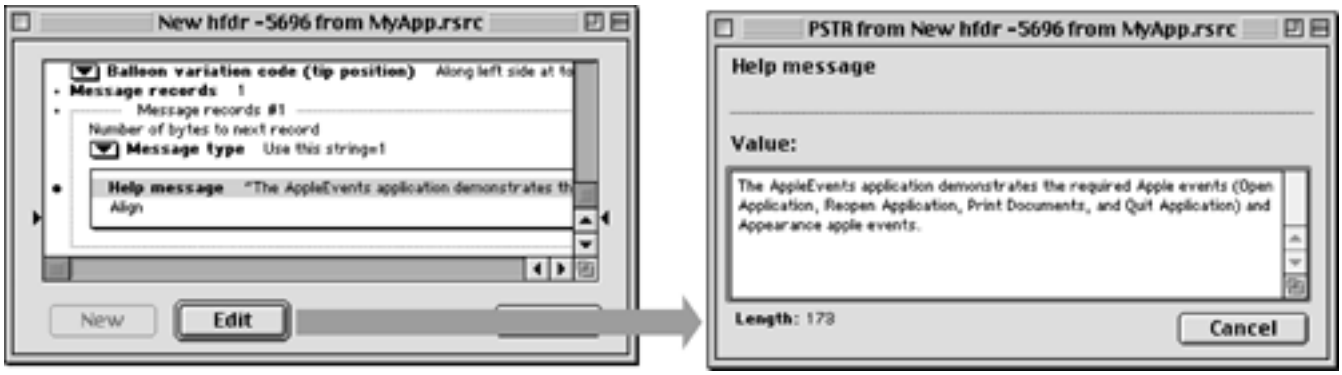
VERSION NUMBER	The version number. This string appears in the Version column in list view windows when the Version checkbox in the View Options dialog is checked.
VERSION MESSAGE	The version number and a company copyright. This string appears in Get Info/General Information windows.

Finder Icon Help ('hfdr') Resource-Mac OS 8/9

The Mac OS 8/9 Finder provides a default help balloon for your application icon. However, you can provide a customised help balloon by adding a finder icon help override (`'hfdr'`) resource with resource ID -5696 to the resource fork of your application.

Fig 9 shows a `'hfdr'` resource being created using Resorcerer.

Figure 9. CREATING AN 'hfdr' RESOURCE USING RESORCERER



Missing Application and Application-Missing Resources-Mac OS 8/9

Missing application and application missing string resources are not supported on Mac OS X. Accordingly, the following is applicable to Mac OS 8/9 only.

Missing Application and Application Missing

On Mac OS 8/9, if the user tries to open a file created by an application that is missing in circumstances where the user has selected Translate documents automatically off in the File Translation tab of the File Exchange control panel, or if the user attempts to open a file that he/she should not be able to open (such as a Preferences file), the Finder responds by displaying a movable modal alert. The contents of that alert depend on the following factors:

- If the file is a document file created by an application that is missing, whether the resource fork of that file contains a **missing application name string resource,** that is, a `'STR '` resource with a resource ID of -16396.
- If the file is a file that is not meant to be opened, whether the file's resource fork contains an **application-missing string resource,** that is, a `'STR '` resource with ID -16397.

If the Finder cannot find the document's creator on any mounted volume, it looks first for the application-missing string resource. If it cannot find an application-missing string resource, it then searches for a missing-application name string resource.

Missing Application

Assuming that the document's name is "Document A," the alert in Fig 10 is displayed if the file does not contain a missing application name string resource and either the document is not of type `'TEXT'` or `'PICT'` or the document is of type `'TEXT'` or `'PICT'` and SimpleText is not available. If the document is of type `'TEXT'` or `'PICT'` and SimpleText is available, the alert in Fig 11 is displayed.

Figure 10. THE DEFAULT MISSING APPLICATION MODALALERT BOX

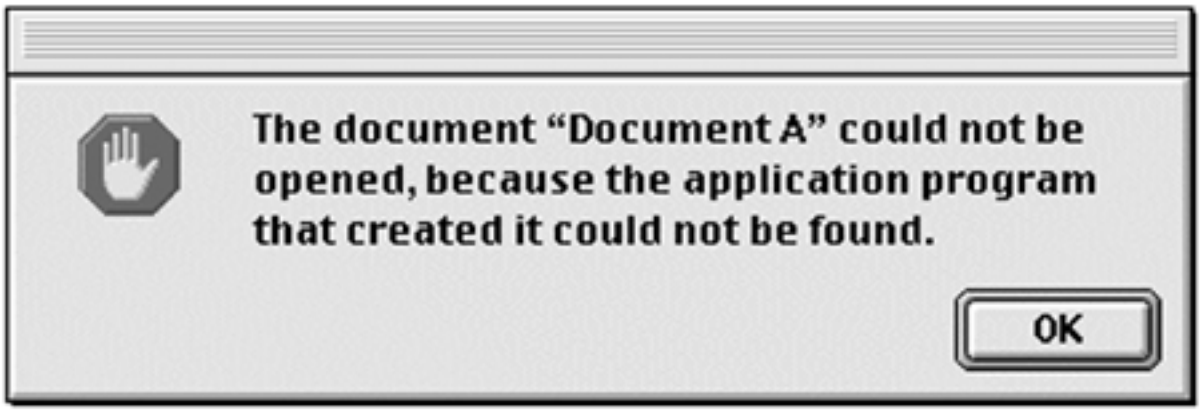
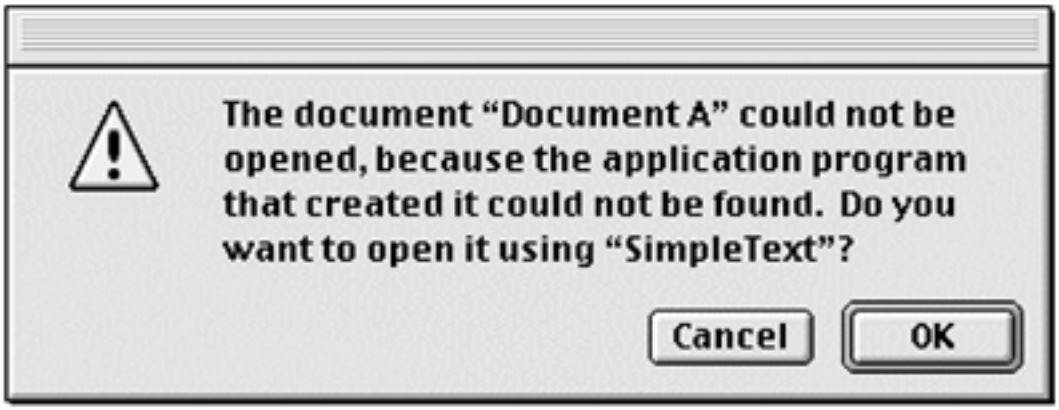


Figure 11. THE MISSING APPLICATION MODAL ALERT BOX - DOCUMENT IS A 'TEXT' OR 'PICT' FILE AND SIMPLETEXT IS AVAILABLE

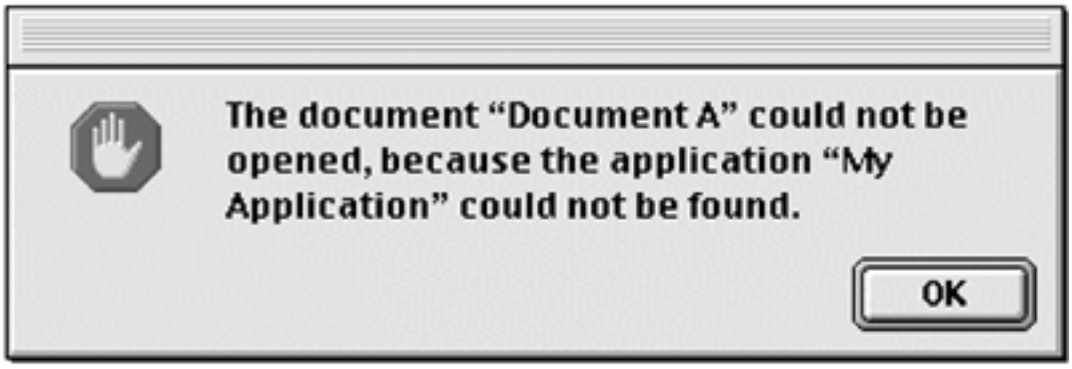

The document “Document A” could not be opened, because the application program that created it could not be found. Do you want to open it using “SimpleText”?
Cancel
OK

Missing Application Name String Resource

The purpose of the missing application name string resource is to provide more specific information to the user. That information is the name of the application that created the program. This is achieved by

- Including a `'STR '` resource containing your application's name in your application's resource fork.
- When your application saves a document for the first time, copying the resource from your application's resource fork to the resource fork of the newly created document, ensuring that its resource ID is -16396. If this resource is present in the document's resource fork, and the user attempts to open the document when your application is not present, a movable modal alert similar to that in Fig 12 is displayed.

Figure 12. THE MISSING APPLICATION MODAL ALERT BOX - DOCUMENT HAS A MISSING APPLICATION NAME STRING RESOURCE



Application-Missing

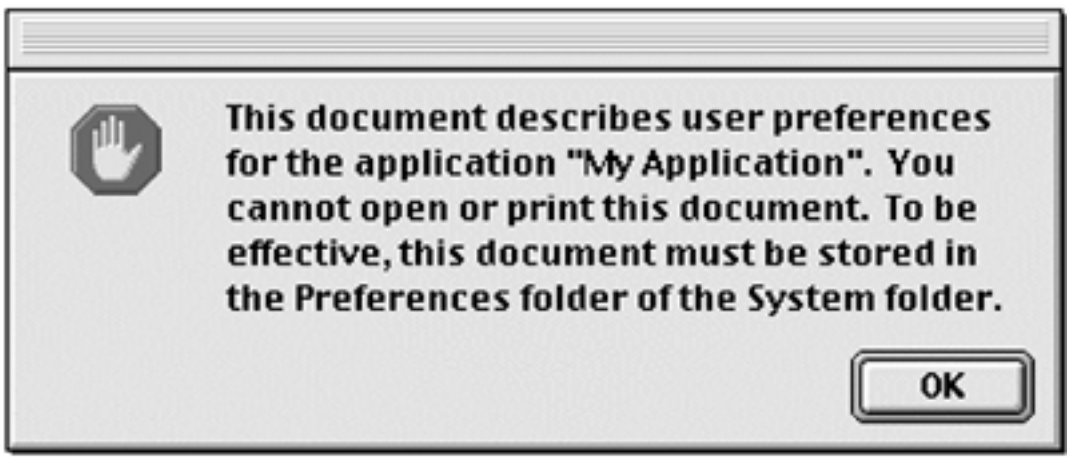
If the user attempts to open a file that is not meant to be opened, and if the file does not contain an application-missing string resource, the Finder displays a modal alert similar to that in Fig 10.

Application-Missing String Resource

The purpose of the application-missing resource is to provide more specific information to the user. That information is an explanation of why the file cannot be opened. This is achieved by

- Including a `'STR '` resource containing the explanation in your application's resource fork.
- When your application creates the file, copying the resource from your application's resource fork to the resource fork of the file, ensuring that its resource ID is -16397. If this resource is present in the document's resource fork, and the user attempts to open the file, a movable modal alert similar to that in Fig 13 is displayed.

Figure 13. AN APPLICATION MISSING MODALALERT BOX DISPLAYING A TYPICAL APPLICATION-MISSING STRING RESOURCE


This document describes user preferences for the application "My Application". You cannot open or print this document. To be effective, this document must be stored in the Preferences folder of the System folder.
OK

The file must have a unique signature that no application known to the finder is likely to have. This ensures that the Finder will display your message, instead of attempting to launch the application with that signature, when the user double clicks on the file's icon.

One 'STR ' Resource Only

You supply *either* the missing-application name string resource *or* the application-missing string resource, but never both. For example, you would supply an application-missing string resource for your Preferences file (which the user should not be able to open) and a missing-application name string resource for documents that users should be able to open with your application.

The 'plst' Resource and Information Property Lists

As previously stated:

- If your application includes a 'plst' (property list) resource, which itself includes a binary representation of an information property list containing the necessary information, the Mac OS X Finder extracts the information it requires from that list. In essence, the information property list is the interface through which your application communicates essential data to the Finder.
- If the 'plst' resource is empty, the Mac OS X Finder extracts the information provided by the signature, file reference, bundle, and version resources described in this chapter.

An information property list is a textual method of representing data that uses the Extensible Markup Language (XML) as the structuring medium, and which uses predefined keys for specifying information of interest to the Finder. This information is specified as key-value pairs (that is, as a dictionary). Standard keys are defined by Mac OS X for information such as the version string to be displayed in the Finder's Show Info dialog and the Finder defines keys for information such as the application's signature and type definitions for document types the application understands.

Example Information Property List

The following is an example of an information property list for a simple non-bundled application which edits text files:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist SYSTEM "file://localhost/System/Library/DTDs/PropertyList.dtd">
<plist version="0.9">
<dict>
  <key>CFBundleInfoDictionaryVersion</key>
  <string>6.0</string>
  <key>CFBundleIdentifier</key>
  <string>com.Smith Software.MyApp</string>
  <key>CFBundleShortVersionString</key>
  <string>1.0.0</string>
  <key>CFBundleDevelopmentRegion</key>
  <string>English</string>
  <key>NSHumanReadableCopyright</key>
  <string>Copyright © 2001 Smith Software</string>
  <key>NSAppleScriptEnabled</key>
  <string>NO</string>
  <key>CFBundleName</key>
  <string>MyApp</string>
  <key>CFBundlePackageType</key>
  <string>APPL</string>
  <key<CFBundleSignature></key>
  <string>myaP</string>
  <key>CFBundleIconFile</key>
  <string>128</string>
  <key>CFBundleShortVersionString</key>
  <string>Version 1.0.0 May 2001</string>
  <key>CFBundleDocumentTypes</key>
  <array>
    <dict>
```

```
      <key>CFBundleTypeIconFile</key>
      <string>129</string>
      <key>CFBundleTypeName</key>
      <string>MyApp TEXT Document</string>
      <key>CFBundleTypeOSTypes</key>
      <array>
        <string>TEXT</string>
      </array>
      <key>CFBundleTypeRole</key>
      <string>Editor</string>
    </dict>
  </array>
</dict>
</plst>
```

The following describes the standard and Finder keys and associated values in this information property list:

Standard Keys	***Value***
`CFBundleInfoDictionaryVersion`	A string used to support future versioning of the format.
`CFBundleIdentifier`	A string, used by the preferences system to uniquely identify the application, in the form of a Java-style package name.
`CFBundleVersion`	A string containing a `'vers'` resource-style (see Fig 8) version number.
`CFBundleDevelopmentRegion`	A string specifying the application's "native" region. In `'plst'` resources, this serves the same purpose as the Region Code in a `'vers'` resource.
`NSHumanReadableCopyright`	A string containing copyright information to be displayed in the Finder's Show Inspector dialog.
`NSAppleScriptEnabled`	A string specifying whether the application is AppleScript enabled.

Finder Keys	***Value***
`CFBundleName`	A string containing the name of the application.
`CFBundlePackageType`	A string containing, for applications, the file type `APPL`.
`CFBundleSignature`	A string containing the application's four-letter creator code.
`CFBundleIconFile`	A string containing the resource ID of the `'icns'` resource containing the application's icon.
`CFBundleShortVersionString`	A string containing human-readable description of the version. This should be more than just the string that can be generated from the `CFBundleVersion` key.
`CFBundleDocumentTypes`	An array of type definitions for document types the application understands.
`CFBundleTypeName`	A string containing a name for this document type.
`CFBundleTypeIconFile`	A string containing the resource ID of the `'icns'` resource containing this file type's icon.

`CFBundleTypeRole`	A string defining the application's role in relation to this document type. The role can be `Editor`, `Viewer`, or `None`.
`CFBundleTypeOSTypes`	An array of strings containing the four letter file type codes that map to this type.

Creating the 'plst' Resource

You can create the complete `'plst'` resource by simply copying the XML listing and pasting it into an empty `'plst'` resource using Resorcerer.

Application Launching by the Finder

The following describes how the Finder launches an application.

When the user double clicks your application icon, selects it and chooses Open from the Finder's File menu, or chooses it from the Mac OS 8/9 Apple menu, the Finder calls the Process Manager to launch your application. The Finder then sends your application a required Apple event (called an Open Application event) before relinquishing control to your application. In response to the Open Application event, your application should then perform its usual startup actions, such as opening an untitled document window.

When the user double-clicks on the icon for a document created by your application, or selects one or more document icons and chooses Open from the Finder's File menu or chooses Print from the Mac OS 8/9 File menu, the Finder reads the creator field of each selected file to find the document's creator. If a document's creator matches your application's signature, the Finder calls the Process Manager to launch your application, and then sends your application a required Apple event (called an Open Documents or (Mac OS 8/9 only) Print Documents event) before relinquishing control to your application. These events contain the name, or names, of the document, or documents, to open or (on Mac OS 8/9 only) print. Your application should then open the documents or print them, as appropriate.

When the user drags one or more document icons to your application's icon, the Finder determines whether to launch your application by comparing the document's file type (stored in the catalog file) with the list of the file types supported by your application. If the document's type appears in your application's `'FREF'` resource or, where your application's `'plst'` resource contains an information property list, in that information property list (Mac OS X only), the Finder calls the Process Manager to launch your application, and passes the name/s of the document/s to your application in a required Apple event (Open Documents event). Your application should then open the document.

If the user double clicks a document item while your application is already running, the Finder sends your application an Open Documents event.

The Finder and the Catalog File

The catalog file, which maintains relationships between the files and directories on a volume, is of prime importance to the File Manager; however, information in the catalog file is also used by the Finder.

In the catalog file, information for files is stored in **file information** (`FInfo`) and **extended file information** (`FXInfo`) **structures** and information for directories is stored in **directory information** (`DInfo`) and **extended directory information** (`DXInfo`) structures. The Finder can manipulate fields in file information, directory information, and extended directory information structures.

When your application creates a new file, it assigns file type and creator information to the relevant fields of the file's file information structure, see Chapter 18. The Finder manipulates the other fields of the file information structure.

The file information structure is as follows:

```
struct FInfo
{
  OSType fdType;     // Type of file.
  OSType fdCreator;  // File's creator.
  UInt16 fdFlags;    // Finder flags (invisible, locked, stationery, etc).
  Point  fdLocation; // File's location in folder. ({ 0,0}  places item automatically.)
  SInt16 fdFldr;     // Folder containing file.
} ;
typedef struct FInfo FInfo;
```

For a file that has already been created, you can set the type, creator and flags fields using the File Manager function `FspSetFInfo`, having first called `FSpGetInfo` to return the file information structure.

Finder Flags

Individual bits of the `fdFlags` field of the file information structure may be examined and set using constants defined in the header file Finder.h. The bits of most interest to an application, and the associated constants, are as follows:

Bit	*Constant*	*Meaning When Set*
10	`kHasCustomIcon`	The file has a customised icon.
11	`kIsStationery`	The file is a stationery pad. To support stationery pads, your application should check this bit for every document passed to it by the Finder and Navigation Services.
12	`kNameLocked`	The file cannot be renamed from the Finder (and, also, cannot have customised icons assigned to it).
13	`kHasBundle`	The file has a `'BNDL'` resource. If the `hasBundle` bit is set, the Finder checks the `hasBeenInited` flag. If the `hasBeenInited` bit is set, the Finder uses database information to display the file's icon. If the `hasBeenInited` bit is not set, the Finder copies the information from the bundle resource to the database and sets the `hasBeenInited` bit. (If the `hasBundle` bit is not set, the Finder displays the default icon for that file type.)

14	`kIsInvisible`	The file is invisible from the Finder and from Navigation Services.
15	`kIsAlias`	The file is an alias.

Preferences, Temporary Items, and Trash Folders-Using FindFolder

The only system-related folders you are ever likely to need are the Preferences folder, the Temporary Items folder, and the Trash folder.

You can use the `FindFolder` function to get the volume reference number and directory ID of these folders. The following **folder type constants** are passed in the `folderType` parameter of `FindFolder` to specify the folder type:

Constant	*Value*	*Folder*
`kPreferencesFolderType`	`'pref'`	Preferences
`kTemporaryFolderType`	`'temp'`	Temporary items
`kTrashFolderType`	`'trsh'`	Trash

Stationery Pads

Stationery Pads and the Finder

If the user opens a stationery pad from the Finder, the Finder first checks the `isStationeryAware` bit in your application's `'SIZE'` resource. If the `isStationeryAware` bit is set, the Finder informs your application that the user has opened a document and passes your application the name of the stationery pad. If the `isStationeryAware` bit is not set, the Finder *creates a new document* from the template and starts up your application, passing it the name of the *new* document.

Stationery Pads and Navigation Services

Unlike the Finder, Navigation Services (see Chapter 18) always passes your application the stationery pad itself, not a copy of it, regardless of the setting of the `isStationeryAware` bit in your application's `'SIZE'` resource. This means that the user can mistakenly over-write the stationery pad by saving it without assigning it a new name, a danger that can be avoided by making your application stationery-aware.

The key step in making your application stationery-aware is to always checking the `kIsStationery` bit of a document before opening it. The following is an example function that takes a file system specification structure and returns `true` or `false` according to whether the file is a stationery document or not:

```
Boolean  isStationeryDoc(FSSpec fileSSpec)
{
  OSErr   osErr;
  FInfo   fInfo;
  Boolean result;

  osErr = FSpGetFInfo(&fileSSpec,&fInfo);
  if(osErr == noErr)
    result = ((fInfo.fdFlags & kIsStationary) == kIsStationary);
  else
    result = false;

  return(result);
}
```

Using Aliases

An **alias** is an object that represents another file, directory, or volume. Both the Finder and Navigation Services resolve aliases before passing them to your application. However, if your application opens a file or directory without going through the Finder or Navigation Services, your application should always call `ResolveAlias` just before opening the file.

Mac OS 9 Packages and Mac OS X Bundles

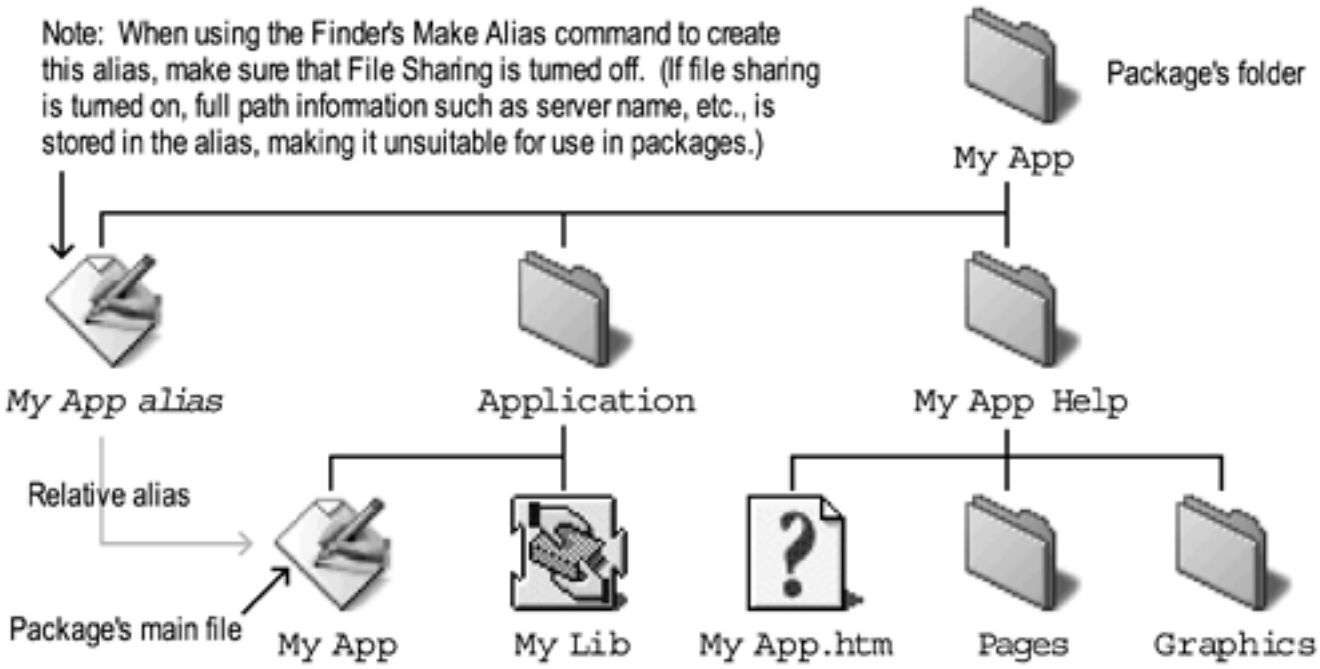
Mac OS 9 Packages

Mac OS 9 introduced **packages.** Basically, a package is a folder with the "package bit" set. The package bit is bit 13 of the `frFlags` field of the directory information structure. This bit is the equivalent, for folders, of the bundle bit for files (bit 13 of the `fdFlags` field of the file information structure. Recall that the bundle bit for files determines how the file's icon is displayed.)

Any folder that has the package bit set, and which contains exactly one alias at its topmost level, will be treated by the Finder as a package.

Normally, a package comprises an application and a number of support files organized into sub-directories. There are no rules governing the arrangement of files and sub-directories other than there must be one alias at the topmost level. This alias points to some other file in the package's directory hierarchy. The file pointed to is called the "package's main file," and is ordinarily an application. An example of an application package is shown in Fig 14.

Figure 14. A PACKAGE(EXAMPLE)


Note: When using the Finder's Make Alias command to create this alias, make sure that File Sharing is turned off. (If file sharing is turned on, full path information such as server name, etc., is stored in the alias, making it unsuitable for use in packages.)
Package's folder
My App
My App alias
Application
My App Help
Relative alias
Package's main file
My App
My Lib
My App.htm
Pages
Graphics

The Mac OS 8/9 Finder treats this package as if it were a file containing the `'BNDL'`, `'FREF'`, icon resources, etc., found in the package's main file. The package's folder is visually presented to the user as if it were a single double-clickable file, specifically, the package's main file. If,

for example, a file is dragged over an application package's icon, the Finder will track the drag command as if the file was being dragged over the package's main file.

The major advantages of application packages are as follows:

- Since they cause the Finder to hide the contents of a package's folder from the user, they prevent accidental (or intentional) tampering with an application's support files.
- They simplify installation and uninstallation. All the user has to do is drag the application package onto a volume or, for uninstall, drag it to the trash.

Note also that the package's name and the package's main file name may differ. This allows the user to edit the name of the package without disturbing the name of the package's main file.

Creating Packages

The Apple application PackageTool may be used to convert correctly formatted folders into packages and vice versa.

Mac OS X Bundles

On Mac OS X, the option exists to package an application as a **bundle.** A bundle is a directory that contains, in a hierarchical organization, the application's executable and the resources required to support that code. One of the required contents of a bundle directory is a file containing the same XML code as would be contained in the `'plst'` resource in a non-bundled version of the application.

A bundle can be presented to the user as a directory. However, as is the case with Mac OS 9 packages, if bit 13 of the `frFlags` field of the directory information structure is set, the Finder presents the bundle directory to the user as if it were a single file.

Packaging applications as bundles is not addressed in this book.

Relevant Constants, Data Types, and Functions

Constants

Finder Flags

```
kIsOnDesk              = 0x1
kColor                 = 0xE
kIsShared              = 0x40
kHasBeenInited         = 0x100
kHasCustomIcon         = 0x400
kIsStationary          = 0x800
kNameLocked            = 0x1000
kHasBundle             = 0x2000
kIsInvisible           = 0x4000
kIsAlias               = 0x8000
```

Folder Type Constants

```
kTrashFolderType       = FOUR_CHAR_CODE('trsh')
kPreferencesFolderType = FOUR_CHAR_CODE('pref')
kTemporaryFolderType   = FOUR_CHAR_CODE('temp')
```

Data Types

File Information Structure

```
struct FInfo
{
  OSType fdType;     // Type of file.
  OSType fdCreator;  // File's creator.
  UInt16 fdFlags;    // Finder flags (invisible, locked, stationery, etc).
  Point  fdLocation; // File's location in folder. ({ 0,0}  places item automatically.)
  SInt16 fdFldr;     // Folder containing file.
} ;
typedef struct FInfo FInfo;
```

Directory Information Structure

```
struct DInfo
{
  Rect   frRect;      // Folder's window bounds.
  Uint16 frFlags;     // Finder flags (invisible, locked, etc).
  Point  frLocation;  // Folder location. ({ 0,0}  places item automatically.)
  SInt16 frView;      // Reserved. (Set to 0.)
} ;
typedef struct DInfo DInfo;
```

Functions

```
OSErr  FSpGetFInfo(const FSSpec *spec,FInfo *fndrInfo);
OSErr  FSpSetFInfo(const FSSpec *spec,const FInfo *fndrInfo);
OSErr  ResolveAliasFile(FSSpec *theSpec,Boolean resolveAliasChains,Boolean *targetIsFolder,
       Boolean *wasAliased);
OSErr  FindFolder(short vRefNum,OSType folderType,Boolean createFolder,short *foundVRefNum,
       long *foundDirID);
```

Demonstration Programs

As previously stated, this chapter should be read in conjunction with Chapter 10. The resources for the demonstration program in this chapter include a signature resource, icon resources, `'FREF'` resources, a `'BNDL'` resource, a `'vers'` resource, a `'hfdr'` resource, and a `'plst'` resource containing an information property list.

Missing Application Name String and Application-Missing String ('STR ') Resources

The demonstration program in Chapter 18 shows how to copy a missing application name string resource to the resource fork of a document file.

The demonstration program in Chapter 19 shows how to copy an application-missing string resource to the resource fork of a Preferences file.

Preferences, Temporary Items, and Trash Folders

The demonstration program in Chapter 19 demonstrates creating and accessing a preferences file in the Preferences folder.

The demonstration program in Chapter 18 demonstrates creating a temporary file in, and deleting it from, the Temporary Items folder.

The demonstration program in Chapter 18 demonstrates determining whether a file is in the Trash folder, or in a folder in the Trash folder.

Chapter 10. Apple Events

IN THIS CHAPTER

- Introduction
- Apple Event Attributes and Parameters
- Data Structures Within Apple Events
- Handling Apple Events
- Required Apple Events-Contents and Required Action
- Main Apple Event Manager and Appearance Manager Constants, Data Types, and Functions Relevant to Required Apple Events and Appearance Manager Apple Events
- Demonstration Program AppleEvents
- Demonstration Program AppleEvents Comments

Introduction

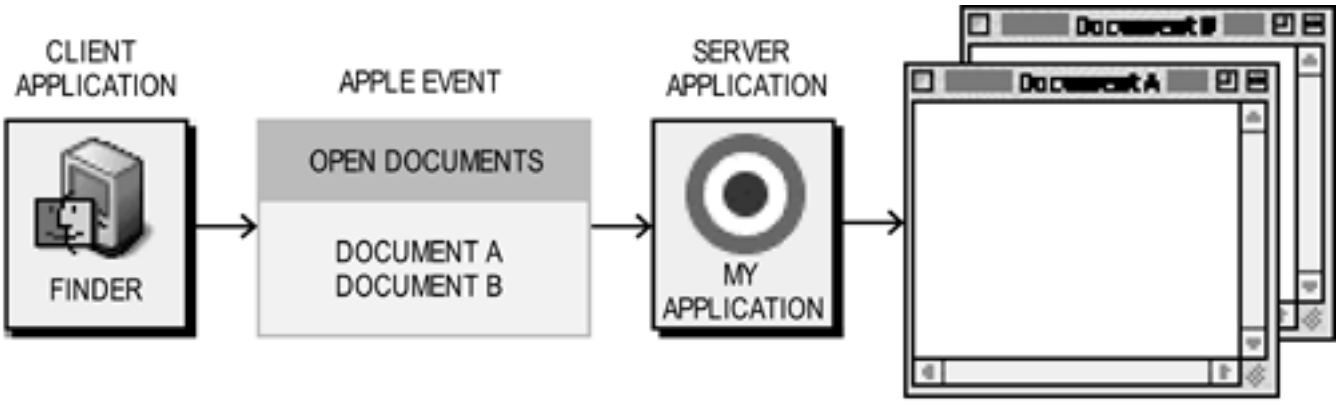
Apple events are **high-level events** whose structure and interpretation are determined by the **Apple Event InterProcess Messaging Protocol** (AEIMP). Applications typically use Apple events to request services and information from other applications and to provide services and information in response to such requests.

In the world of Apple events

- The **client application** is the application that initiates communication between two applications that support Apple events. It sends the Apple event that requests a service or information.
- The **server application** is the application that provides the requested service or information.

As an example, Fig 1 shows a client application (the Finder) sending an Apple event known as the Open Documents event to the server application (My Application) requesting the latter to open the documents named Document A and Document B. My Application responds by opening windows for the specified documents.

Figure 1. CLIENT AND SERVER



An application can also send Apple events to itself, thus acting as both client and server.

Applications can rely on a vocabulary of standard Apple events, defined in the Apple Event Registry: standard suites, to identify Apple events and respond appropriately. The standard **suites** (a suite being a group of Apple events that are usually implemented together) include:

- The **required** suite, which consists of five Apple events that the Finder sends to applications. The required Apple events are:

    - Open Application.
    - Re-open Application.
    - Open Documents.
    - Print Documents.
    - Quit Application.

  The Finder uses the required Apple events as part of the mechanism for launching and terminating applications. Your application must support the required Apple events.

- The **core** suite, which consists of the basic Apple events, including Get Data, Set Data, Move, Delete and Save, that nearly all applications use to communicate.
- The **functional-area** suite, which consists of a group of Apple events that support a related functional area, and which include the Text suite and the Database suite.
- The **appearance** suite, which pertains to Mac OS 8/9 only, and which consists of four Apple events used to advise all foreground applications when one of the following changes has been made in the Mac OS 8/9 Appearance control panel:

    - Appearance. (Since the only Appearance ever made available by Apple is Platinum, this event will not be considered further.)
    - Large system font.
    - Small system font.
    - Views font.

Another important Apple event, of relevance on Mac OS X only, is the Show Preferences event. This event is sent to your application when the user chooses the Preferences item in the Mac OS X Application menu.

This chapter is primarily concerned with the required Apple events, the Appearance Manager Apple events, and the Show Preferences event, exploring the subject of Apple events only to the extent necessary to gain an understanding of the measures involved in supporting these events.

Apple Event Attributes and Parameters

An Apple event comprises attributes and, possibly, parameters**.** Attributes identify the Apple event and denote the task to be performed. Parameters contain information to be used by the target application.

Apple Event Attributes

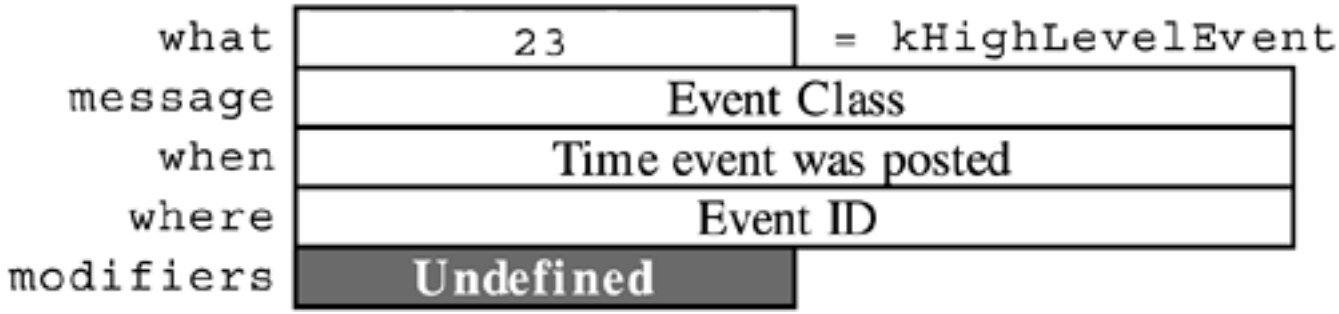
An Apple event attribute is a structure that identifies, principally, the **event class, event ID,** and target application.

Event Class

The event class identifies a group of related Apple events. It appears in the `message` field of the event structure for an Apple event (see Fig 2). For example

- The required Apple events have the value `'aevt'` in the `message` field of their event structures. (`'aevt'` is represented by the constant `kCoreEventClass`.)
- The Appearance Manager Apple events have the value `'appr'` in the `message` field of their event structures. (`'appr'` is represented by the constant `kAppearanceEventClass`.)

Figure 2. CONTENTS OF AN EVENT STRUCTURE-HIGH LEVEL (APPLE) EVENT



Event ID

The event ID identifies the particular event within the event class, uniquely identifying the Apple event and communicating the action the Apple event should perform. As shown in Fig 2, the event ID appears in the `where` field of the event structure for an Apple event. For example, for an Open Documents event, the where field will contain the value `'odoc'` (which is represented by the constant `kAEOpenDocuments`.)

The following are the event IDs for the five required Apple events and the four Appearance Manager Apple events.

Event ID	*Value*	*Description*
`kAEOpenApplication`	`'oapp'`	Perform those tasks associated with the user opening the application.
`kAEReopenApplication`	`'rapp'`	Perform those tasks associated with the user "re-opening" the application.
`kAEOpenDocuments`	`'odoc'`	Open documents.
`kAEPrintDocuments`	`'pdoc'`	Print documents.
`kAEQuitApplication`	`'quit'`	Quit the application.
`kAEAppearanceChanged`	`'thme'`	Current appearance has changed. Action as required.
`kAESystemFontChanged`	`'sysf'`	Current system font has changed. Action as required.
`kAESmallSystemFontChanged`	`'ssfn'`	Current small system font has changed. Action as required.
`kAEViewsFontChanged`	`'vfnt'`	Current views font has changed. Action as required.
`kAEShowPreferences`	`'pref'`	User has chosen the Prefences item in the Mac OS X Application menu.

Target Application

In addition to the event class and event ID, every Apple event must include an attribute that specifies the target application's address.

Apple Event Parameters

An Apple event parameter is a structure containing data used by the target application. Apple events can use standard data types, such as strings of text, long integers, boolean values, and alias structures, for the data in their parameters.

There are various kinds of Apple event parameters, including direct parameters and additional parameters.

Direct Parameters

Direct parameters usually specify the data to be acted upon by the target application. For example, a list of documents is contained in the direct parameter of the Open Documents event.

Additional Parameters

Some Apple events also take additional parameters. The target application uses these additional parameters (for example, operands in an equation) in addition to the data specified in the direct parameter.

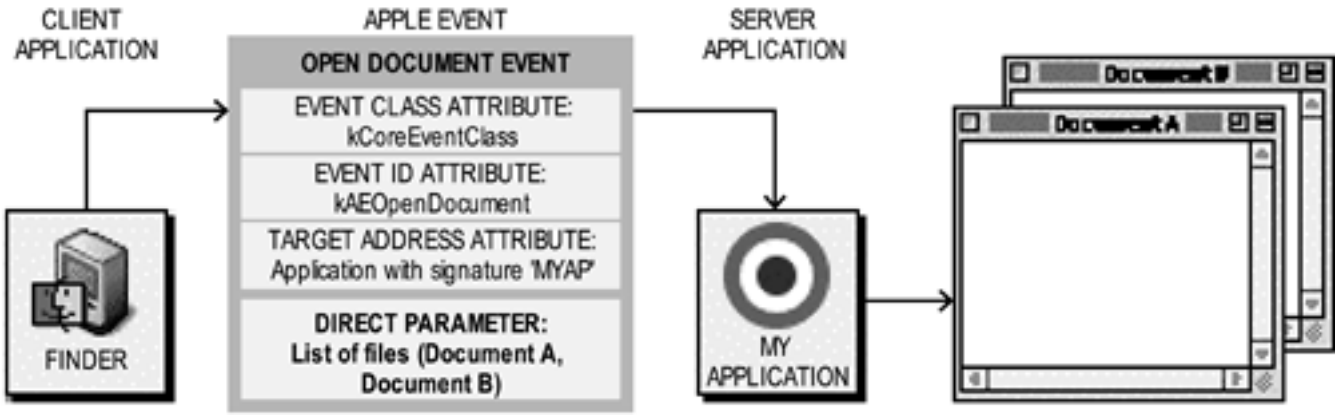
Required and Optional Parameters

All parameters are described as either **required parameters** or **optional parameters** in the Apple Event Registry: Standard Suites. Direct parameters are usually defined as required parameters.

Attributes and Parameters in an Open Documents Apple Event

Fig 3 shows the major Apple event attributes and the direct parameter for the Open Documents event.

Figure 3. OPEN DOCUMENTS APPLE EVENT - MAJOR ATTRIBUTES AND DIRECT PARAMETER



To process this event, the application My Application would use the `AEProcessAppleEvent` function, which uses the event class and event ID attributes to dispatch the event to its Open Documents handler. The Open Documents handler opens the documents specified in the direct parameter.

Data Structures Within Apple Events

The Apple Event Manager constructs its own internal data structures to contain the information in an Apple event.

Descriptor Structures

The Apple Event Manager uses **descriptor structures,** which you can think of as building blocks, to construct Apple event attributes and parameters. Descriptor structures comprise a handle to data and a **descriptor type.** The descriptor type identifies the type of data to which the handle refers:

```
struct AEDesc
{
  DescType      descriptorType;  // Type of data.
  AEDataStorage dataHandle;      // Handle to data.
} ;
typedef struct AEDesc AEDesc;
```

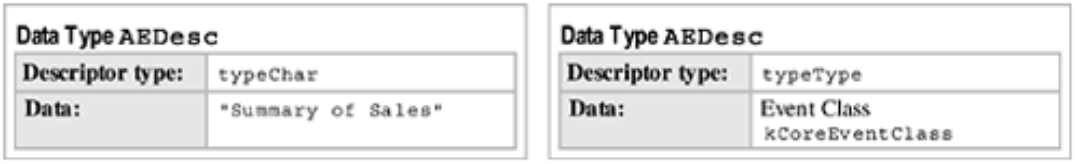
> **Carbon Note**
>
> In Carbon, the `dataHandle` field is opaque. You must use the accessor functions `AEGetDescData` and `AEGetDescDataSize` to access the data in this field.

The descriptor type is of type `DescType`, that is, a four-character code. The following are the codes for some of the major descriptor types, the constants used to represent them, and the kind of data they identify:

Descriptor Type	*Value*	*Description of Data*
`typeChar`	`'TEXT'`	Unterminated string.
`typeType`	`'type'`	Four-character code.
`typeBoolean`	`'bool'`	One-byte Boolean value.
`typeLongInteger`	`'long'`	32-bit integer.
`typeAEList`	`'list'`	List of descriptor structures.
`typeAERecord`	`'reco'`	List of keyword-specified descriptor structures.
`typeAppleEvent`	`'aevt'`	Apple event structure.
`typeFSS`	`'fss '`	File system specification.
`typeKeyword`	`'keyw'`	Apple event keyword.
`typeNull`	`'null'`	Nonexistent data (handle whose value is `NULL`).

Fig 4 illustrates the logical arrangement of two descriptor structures. The first specifies that the data handle refers to an unterminated string. The second specifies that the data handle refers to a four-character code, in this case `'aevt'` (which is represented by the constant `kCoreEventClass).`

Figure 4. TYPICAL DESCRIPTOR STRUCTURES



Address Descriptor Structures

Every Apple event includes an attribute specifying the address of the target application. A descriptor structure that contains an application's address is called an **address descriptor structure:**

```
typedef AEDesc AEAddressDesc;  // An AEDesc which contains addressing data.
```

Keyword-Specified Descriptor Structures

The Apple Event Manager assembles the various descriptor structures into an Apple event. Your application cannot access the contents of the Apple event directly; rather, Event Manager functions must be used to request each attribute and parameter by **keyword.** Keywords, which are four-character codes of type `AEKeyword`, are used to keep track of various descriptor structures.

The following are typical keywords and the constants used to represent them:

Keyword	*Value*	*Description*
`keyMissedKeywordAttr`	`'miss'`	For first required parameter remaining in an Apple event.
`keyDirectObject`	`'----'`	Direct parameter.

Keywords are associated with specific descriptor structures by means of keyword-specified descriptor structures:

```
struct AEKeyDesc
{
  AEKeyword descKey;      // Keyword.
  AEDesc    descContent;  // Descriptor structure.
} ;
typedef struct AEKeyDesc AEKeyDesc;
```

Descriptor Lists, AE Structures, and AppleEvent Structures

Descriptor Lists

To extract data from an Apple event, you use Apple Event Manager functions to either copy data to a buffer, return a descriptor structure whose data handle refers to a copy of the data, or return **descriptor lists,** which are lists of descriptor structures. A descriptor list is a descriptor structure whose handle refers to a list of other descriptor structures (unless it is an empty list):

```
typedef AEDesc AEDescList;  // List of descriptor structures.
```

Fig 5 illustrates the logical arrangement of the descriptor list that specifies the direct parameter of the Open Documents event shown in Fig 3.

Figure 5. A DESCRIPTOR LIST FOR A LIST OF ALIASES

Data Type `AEDescList`

Descriptor type:	`typeAEList`
Data:	List of descriptor records

Descriptor type:	`typeAlias`
Data:	Alias record for filename (Document A)

Descriptor type:	`typeAlias`
Data:	Alias record for filename (Document B)

This descriptor list provides the data for a keyword-specified descriptor structure.

AE Structure

Keyword-specified descriptor structures can in turn be combined into an **AE structure.** An AE structure is a descriptor list of type `AERecord`:

```
typedef AEDescList AERecord;  // List of keyword-specified descriptor structures.
```

The handle for a descriptor list of type `AERecord` refers to a list of keyword-specified descriptor structures that can be used to construct Apple event parameters.

Apple Event Structure

An **Apple event structure** is another special descriptor list of type `AppleEvent:`

```
typedef AERecord AppleEvent;  // List of attributes and parameters for Apple event.
```

An Apple event structure describes an Apple event. The data for an Apple event structure, like the data for an AE structure, consists of a list of keyword-specified descriptor structures. The difference between an AE structure and an Apple event structure is that the data in the latter is divided into two parts, the first for attributes and the second for parameters.

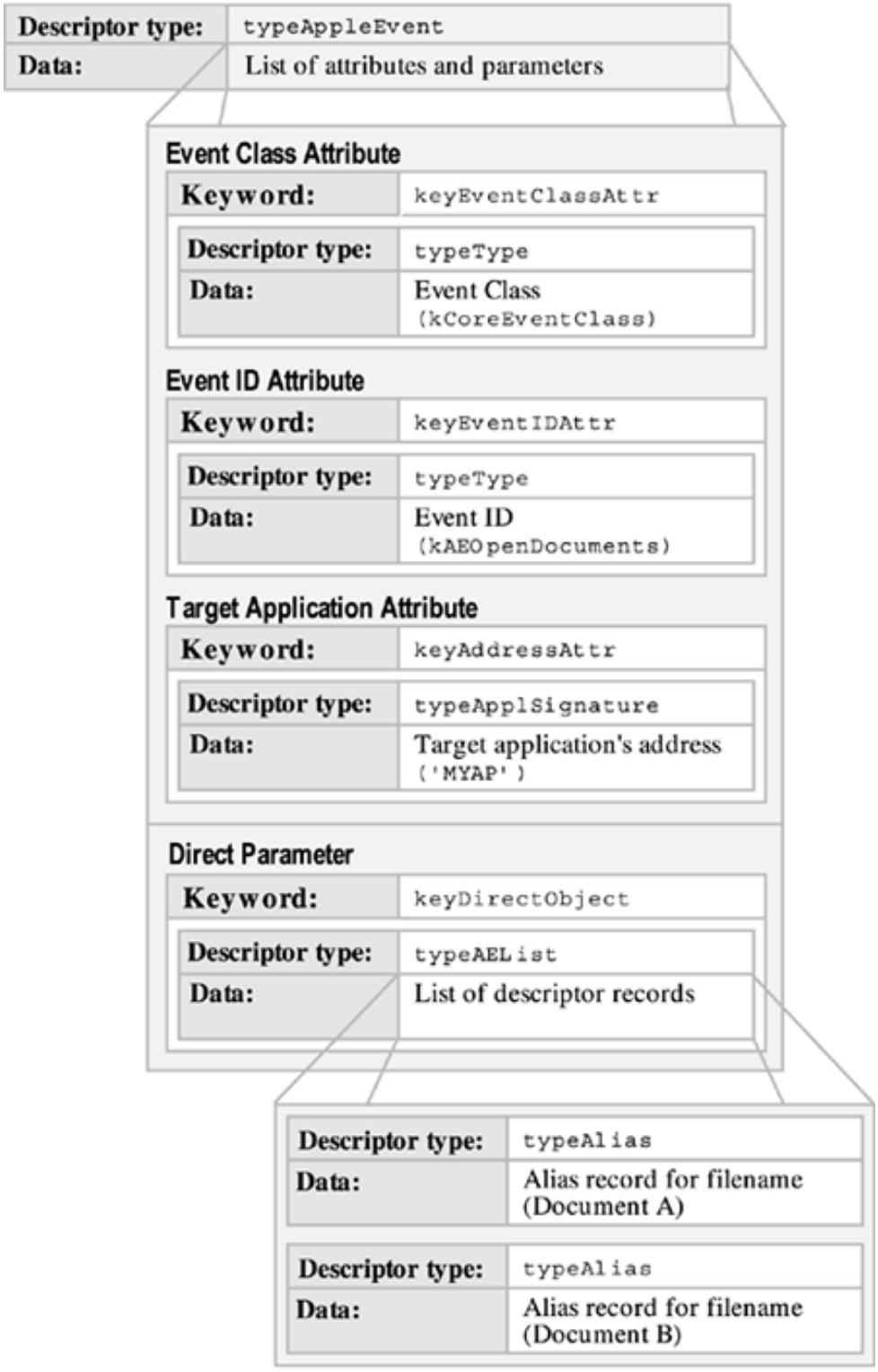
Passing Descriptor Lists, AE Structures, and Apple Event Structures to Apple Event Manager Functions

You can pass an Apple event structure to any Apple Event Manager function that expects an AE structure, and you can pass Apple event structures and AE structures, as well as descriptor lists and descriptor structures, to any Apple Event Manager functions that expect structures of data type `AEDesc`.

Example Complete Apple Event

Fig 6 shows an example of a complete Apple event. This is a data structure of type `AppleEvent` which contains a list of keyword-specified descriptor structures containing the attributes and parameters of an Open Documents event.

Figure 6. AN APPLE (OPEN DOCUMENTS) EVENT


Descriptor type: typeAppleEvent
Data: List of attributes and parameters
Event Class Attribute
Keyword: keyEventClassAttr
Descriptor type: typeType
Data: Event Class (kCoreEventClass)
Event ID Attribute
Keyword: keyEventIDAttr
Descriptor type: typeType
Data: Event ID (kAEOpenDocuments)
Target Application Attribute
Keyword: keyAddressAttr
Descriptor type: typeApplSignature
Data: Target application's address ('MYAP')
Direct Parameter
Keyword: keyDirectObject
Descriptor type: typeAEList
Data: List of descriptor records
Descriptor type: typeAlias
Data: Alias record for filename (Document A)
Descriptor type: typeAlias
Data: Alias record for filename (Document B)

Handling Apple Events

To request a service or information, a client application uses the Apple Event Manager to create and send an Apple event. To respond, a server application uses the Apple Event Manager to extract data from the attributes and parameters of the Apple event. In addition, and where necessary, the server application adds requested data to the reply event returned to the client by the Apple Event Manager.

As previously stated, your application should support the required Apple events sent by the Finder. Your application must therefore

- Set the `isHighLevelEventAware` flag in your application's `'SIZE'` resource.
- Test for high-level events in the event loop. An Apple event (like all high-level events) is identified by a message class of `kHighLevelEvent` in the `what` field of the event structure.
- Use `AEProcessAppleEvent` to process the Apple events. `AEProcessAppleEvent` examines the data in the event class and event ID attributes so as to identify the Apple event and then calls the appropriate **Apple event handler** provided by your application.
- Provide handlers for the required Apple events in your application. Your Apple event handlers must extract the pertinent data from the Apple event, perform the requested action, and return a result.
- Use `AEInstallEventHandler` to install your Apple event handlers in an **Apple event dispatch table** for your application. The Apple event dispatch table is used by the Apple Event Manager to map Apple events to your application's handlers. Calls to `AEProcessAppleEvent` cause the Apple Event Manager to check the dispatch table and, if your application has installed a handler for the event, call the handler.

Apple Event Handlers

Each Apple event handler must be a function that uses this syntax:

```
OSErr theEventHandler(AppleEvent *appleEvent,AppleEvent *reply,long handlerRefcon);
```

`appleEvent` The Apple event to handle. Your handler uses Apple Event Manager functions to extract any parameters and attributes from the Apple event and then perform the necessary processing.

`reply` The default reply provided by the Apple Event Manager.

`handlerRefcon` Reference constant stored in the Apple event dispatch table entry for the Apple event. Your handler can ignore this parameter if your application does not use the reference constant.

Apple event handlers must generally perform the following tasks:

- Extract the attributes and parameters from the Apple event.
- Check that all required parameters have been extracted.
- Perform the action requested by the Apple event.
- Dispose of any copies of the descriptor structures that have been created.
- Add information to the reply Apple event if requested.

Extracting and Checking Data

You must use Apple Event Manager functions to extract the data from Apple events. The following are the main functions

involved:

Function	*Description*
`AEGetAttributePtr`	Uses a buffer to return a copy of the data contained in an Apple event attribute. Used to extract data of fixed length or known maximum length.
`AEGetParamDesc`	Returns a copy of the descriptor structure or descriptor list for an Apple event parameter. Usually used to extract data of variable length, for example, to extract the descriptor list for a list of alias structures specified in the direct parameter of an Open Documents event.
`AECountItems`	Returns the number of descriptor structures in a descriptor list. Used, for example, to determine the number of alias structures for documents specified in the direct parameter of an Open Documents event.
`AEGetNthPtr`	Uses a buffer to return a copy of the data for a descriptor structure contained in a descriptor list. Used to extract data of fixed length or known maximum length, for example, to extract the name and location of a document from the descriptor list specified in the direct parameter of the Open Documents event.

Data Type Coercion

You can specify the descriptor type in the resulting data from these functions. If this type is different from the descriptor type of the attribute or parameter, the Apple Event Manager attempts to coerce it to the specified type. In the direct parameter of the Open Documents event, for example, each descriptor structure in the descriptor list is an **alias structure** and each alias structure specifies a document to be opened. All your application usually needs to open a document is a **file system specification structure** (`FSSpec`) of the document. When you extract the descriptor structure from the descriptor list, you can request that the Apple Event Manager return the data to your application as a file system specification structure instead of an alias structure.

Checking That All Required Parameters Have Been Retrieved

After extracting all known Apple event parameters, your handler should check that it has retrieved all the parameters that the source application considered to be required. To do this, determine whether the `keyMissedKeywordAttr` attribute exists. If this attribute does exist, your handler has not retrieved all the required parameters, and it should return an error.

Performing the Requested Action and Returning a Result

When your application responds to an Apple event, it should perform the standard action requested by the event.

Your Apple event handler should always set its function result to either `noErr`, if it successfully handles the Apple event, or to a non-zero result code if an error occurs. When your handler returns a non-zero result code, the Apple Event Manager adds a `keyErrorNumber` parameter, containing the result code that your handler returns, to the reply Apple event.

Disposing of Copies of Descriptor Structures

When your handler is finished with a copy of a descriptor structure created by `AEGetParamDesc` and related functions, it should dispose of it by calling `AEDisposeDesc`.

Required Apple Events-Contents and Required Action

Your application receives the five required Apple events from the Finder in these circumstances:

- If your application is not open and the user elects to open it from the Finder without opening or printing any documents (either by double clicking the application's icon, selecting the icon and choosing Open from the Finder's File menu, or choosing it from the Mac OS 8/9 Apple menu), the Finder calls the Process Manager to launch your application and then sends your application an Open Application event.
- If your application is already open and the user attempts to "open" it again, the Finder sends your application a Re-open Application event.
- If your application is not open and the user elects to open one of your application's documents from the Finder, the Finder calls the Process Manager to launch your application and then sends your application an Open Documents event.
- On Mac OS 8/9, if your application is not open and the user elects to print one of your application's documents from the Finder, the Finder calls the Process Manager to launch your application and then sends your application a Print Documents event. Your application should print the selected documents and remain open until it receives a Quit Application event from the Finder.
- If your application is open and the user elects to open or (on Mac OS 8/9 only) print any of your application's documents from the Finder, the Finder sends your application the Open Documents or (on Mac OS 8/9 only) Print Documents event.
- If your application is open and the user chooses Restart or Shut Down, the Finder sends your application the Quit Application event.

The Re-open Application event was introduced with MAC OS 8 to cater for a situation which could confuse inexperienced users. The specific situation is where the application is open but has no open windows. Because of the absence of a window, the user does not realise that the application is running, attempts to "open" it from the Finder, and then fails to notice the menu bar change. The intention of the Re-open Application event in such circumstances is to cause the application to open a window, providing more obvious visible evidence to the user that the application is, in fact, open.

The following is a summary of the contents of the required Apple events sent by the Finder and the actions they request applications to perform:

Open Application event

Attributes:

Event Class: `kCoreEventClass`

Event ID: `kAEOpenApplication`

Parameters: None.

Requested Action: Perform tasks your application normally performs when a user opens your application without opening or printing any documents, such as opening an untitled document window.

Re-open Application event

Attributes:

Event Class: `kCoreEventClass`

Event ID: `kAEReopenApplication`

Parameters: None.

Requested Action: If no windows are currently open, open a new untitled document window.

Open Documents event

Attributes:

Event Class: `kCoreEventClass`

Event ID: `kAEOpenDocuments`

Required parameters:

Keyword: `keyDirectObject`

Descriptor type: `typeAEList`

Data: A list of alias structures for the documents to be opened.

Requested Action: Open the documents specified in the `keyDirectObject` parameter.

Print Documents event

Attributes:

Event Class: `kCoreEventClass`

Event ID:	`kAEPrintDocuments`

Required parameters:

Keyword:	`keyDirectObject`
Descriptor type:	`typeAEList`
Data:	A list of alias structures for the documents to be printed.

Requested action: Print the documents specified in the `keyDirectObject` parameter.

Quit Application event

Attributes:

Event Class:	`kCoreEventClass`
Event ID:	`kAEQuitApplication`

Parameters: None

Requested Action: Perform any tasks that your application would normally perform when the user chooses Quit from the application's File menu. (Such tasks typically include releasing memory and asking the user whether to save documents which have been changed.)

Your application needs to recognize two descriptor types to handle the required Apple events: descriptor lists and alias structures.

As previously stated, in the event of an Open Documents or (on Mac OS 8/9 only) Print Documents event, you can retrieve the data that specifies the document as an alias structure, or you can request that the Apple Event Manager coerce the alias structure to a file system specification structure. The file system specification provides a standard method of identifying files.

Main Apple Event Manager and Appearance Manager Constants, Data Types, and Functions Relevant to Required Apple Events and Appearance Manager Apple Events

Constants

High Level Event

```
kHighLevelEvent           = 23
```

Event Classes for Required Apple Event and Appearance Manager Apple Event

```
kCoreEventClass           = FOUR_CHAR_CODE('aevt')  Event class - required Apple events.
KAppearanceEventClass     = FOUR_CHAR_CODE('appr')  Event Class - Appearance Manager Apple
                                                    events.
```

Event IDs for Required Apple Events

```
kAEOpenApplication        = FOUR_CHAR_CODE('oapp')  Event ID for Open Application event.
kAEReopenApplication      = FOUR_CHAR_CODE('rapp')  Event ID for Re-open Application Event.
kAEOpenDocuments          = FOUR_CHAR_CODE('odoc')  Event ID for Open Documents event.
kAEPrintDocuments         = FOUR_CHAR_CODE('pdoc')  Event ID for Print Documents event.
kAEQuitApplication        = FOUR_CHAR_CODE('quit')  Event ID for Quit Application event.
```

Event IDs for Appearance Manager Apple Events

```
kAESystemFontChanged      = FOUR_CHAR_CODE('sysf')  System font changed.
KAESmallSystemFontChanged = FOUR_CHAR_CODE('ssfn')  Small system font changed.
kAEViewsFontChanged       = FOUR_CHAR_CODE('vfnt')  Views font changed.
```

Event ID for Show Preferences Apple Event (Mac OS X)

```
kAEShowPreferences        = FOUR_CHAR_CODE('pref')  Preferences menu item chosen
```

Keywords for Apple Event Attributes

```
keyMissedKeywordAttr      = FOUR_CHAR_CODE('miss')  First required parameter remaining in
                                                    an Apple event.
```

Keywords for Apple Event Parameters

```
keyDirectObject           = FOUR_CHAR_CODE('----')  Direct parameter
```

Apple Event Descriptor Types

```
typeAEList                = FOUR_CHAR_CODE('list')  List of descriptor structures.
typeWildCard              = FOUR_CHAR_CODE('****')  Matches any type.
typeFSS                   = FOUR_CHAR_CODE('fss ')  File system specification.
```

Result Codes

```
errAEDescNotFound         = -1701  Descriptor structure was not found.
errAEParamMissed          = -1715  Handler cannot understand a parameter
                                   the client considers is required.
```

Theme Font ID Constants

```
KThemeSystemFont               = 0
```

```
KThemeSmallSystemFont           = 1
KThemeSmallEmphasizedSystemFont = 2
KThemeViewsFont                 = 3
```

Data Types

```
typedef FourCharCode  AEEventClass;  // Event class for a high level event.
typedef FourCharCode  AEEventID;     // Event ID for a high level event.
typedef FourCharCode  AEKeyword;     // Keyword for a descriptor structure.
typedef ResType       DescType;      // Descriptor type.
typedef AEDesc        AEDescList;    // List of descriptor structures.
typedef AEDescList    AERecord;      // List af keyword-specified descriptor structures.
typedef AERecord      AppleEvent;    // List of attributes and parameters for Apple event.
```

Descriptor Structure

```
struct AEDesc
{
  DescType      descriptorType;  // Type of data being passed.
  AEDataStorage dataHandle;      // Handle to data being passed.
} ;
typedef struct AEDesc AEDesc;
```

Keyword-Specified Descriptor Structure

```
struct AEKeyDesc
{
  AEKeyword deskKey;        // Keyword.
  AEDesc    descContent;    // Descriptor structure.
} ;
typedef struct AEKeyDesc AEKeyDesc;
```

Functions

Creating and Managing Apple Event Dispatch Tables

```
OSErr  AEInstallEventHandler(AEEventClass theAEEventClass,AEEventID theAEEventID,
       AEEventHandlerUPP handler,long handlerRefcon,Boolean isSysHandler);
```

Dispatching Apple Events

```
OSErr  AEProcessAppleEvent(const EventRecord *theEventRecord);
```

Getting Data or Descriptor Structures out of Apple Event Parameters and Attributes

```
OSErr  AEGetParamDesc(const AppleEvent *theAppleEvent,AEKeyword theAEKeyword,
       DescType desiredType,AEDesc *result);
OSErr  AEGetAttributePtr(const AppleEvent *theAppleEvent,AEKeyword theAEKeyword,
       DescType desiredType,DescType *typeCode,Ptr dataPtr,Size maximumSize,
       Size *actualSize);
```

Counting the Items in Descriptor Lists

```
OSErr  AECountItems(const AEDescList *theAEDescList,long *theCount);
```

Getting Items from Descriptor Lists

```
OSErr  AEGetNthPtr(const AEDescList *theAEDescList,long index,DescType desiredType,AEKeyword
       *theAEKeyword,DescType *typeCode,Ptr dataPtr,Size maximumSize,Size *actualSize);
```

Deallocating Memory for Descriptor Structures

```
OSErr  AEDisposeDesc(AEDesc *theAEDesc);
```

Demonstration Program AppleEvents

```
//
//****************************************************************************************
// AppleEvents.c                                                        CLASSIC EVENT
MODEL
//
//****************************************************************************************
//
// This program:
//
// •  Installs handlers for the required Apple events, Appearance Manager Apple events, and,
//    on Mac OS X only, the Show Preferences Apple event.
//
// •  Responds to the receipt of required Apple events by displaying descriptive text in a
//    window opened for that purpose, and by opening simulated document windows as
//    appropriate. These responses result from the user:
//
//    •   Double clicking on the application's icon, or selecting the icon and choosing Open
//        from the Finder's File menu, thus causing the receipt of an Open Application
event.
//
//    •   When the application is already open, double clicking on the application's icon,
or
//        selecting the icon and choosing Open from the Finder's File menu, thus causing the
//        receipt of a Re-open  Application event.
//
//    •   Double clicking on one of the document icons, selecting one or both of the
document
//        icons and choosing Open from the Finder's File menu, or dragging one or both of
the
//        document icons onto the application's icon, thus causing the receipt of an Open
//        Documents event.
//
//    •   On Mac OS 8/9, selecting one or both of the document icons and choosing Print from
//        the Finder's file menu, thus causing the receipt of a Print Documents event and,
if
//        the application is not already running, a subsequent Quit Application event.
//
//    •   While the application is running, choosing Shut Down or Restart from the Finder's
//        Special menu, thus causing the receipt of a Quit Application event.
//
// •  Responds to the receipt of Appearance Manager Apple events (Mac OS 8/9) and the Show
//    Preferences Apple event (Mac OS X) by displaying descriptive text.
//
// The program, which is intended to be run as a built application rather than within
// CodeWarrior, utilises the following resources:
//
// •  A 'plst' resource containing an information property list which provides information
//    to the Mac OS X Finder.
//
// •  An 'icns' resource containing application and document icons for Mac OS X.
//
// •  'WIND' resources (purgeable, initially visible) for the descriptive text display
window
//     and simulated document windows.
//
// •  'MBAR' and 'MENU' resources (preload, non-purgeable).
//
// •  'STR#' resources (purgeable) for displaying error messages using StandardAlert.
//
// •  For Mac OS 8/9:
```

```
//
//    •   'ICN#', 'ics#', 'ics4', 'ics8', 'icl4', and 'icl8' resources (that is, an icon
//        family) for the application and for the application's documents. (Purgeable.)
//
//    •   'FREF' resources (non-purgeable) for the application and the application's 'TEXT'
//        documents, which link the icons with the file types they represent, and which
allow
//        users to launch the application by dragging the document icons to the application
//        icon.
//
//    •   The application's signature resource (non-purgeable), which enables the Finder to
//        identify and start up the application when the user double clicks the
application's
//        document icons.
//
//    •   A 'BNDL' resource (non-purgeable), which groups together the application's
//        signature, icon and 'FREF' resources.
//
//    •   A 'hfdr' resource (purgeable), which provides the customised finder icon help
//        override help balloon for the  application icon.
//
//    •   A 'vers' resource (purgeable), which provides version information via the Show
Info
//        window and the Version column in list view windows.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
//
*****************************************************************************************

// ..................................................................................
includes

#include <Carbon.h>

// ..................................................................................
defines
#define rMenubar             128
#define mFile                129
#define  iQuit               12
#define rDisplayWindow       128
#define rDocWindow           129
#define rErrorStrings        128
#define eInstallHandler      1
#define eGetRequiredParam    2
#define eGetDescriptorRecord 3
#define eMissedRequiredParam 4
#define eCountDescripRecords 5
#define eCannotOpenFile      6
#define eCannotPrintFile     7
#define eCannotOpenWindow    8
#define eMenus               9
#define MIN(a,b)             ((a) < (b) ? (a) : (b))

// ..................................................................... global variables

WindowRef gWindowRef;
Boolean   gRunningOnX = false;
Boolean   gDone;
Boolean   gApplicationWasOpen = false;

// ..................................................................... function prototypes

void      main                          (void);
void      doPreliminaries               (void);
```

```
void      doInstallAEHandlers           (void);
void      doInstallAnAEHandler          (AEEventClass,AEEventID,void *);
void      doEvents                      (EventRecord *);
OSErr     openAppEventHandler           (AppleEvent *,AppleEvent *,SInt32);
OSErr     reopenAppEventHandler         (AppleEvent *,AppleEvent *,SInt32);
OSErr     openDocsEventHandler          (AppleEvent *,AppleEvent *,SInt32);
OSErr     printDocsEventHandler         (AppleEvent *,AppleEvent *,SInt32);
OSErr     quitAppEventHandler           (AppleEvent *,AppleEvent *,SInt32);
OSErr     sysFontChangeEventHandler     (AppleEvent *,AppleEvent *,SInt32);
OSErr     smallSysFontChangeEventHandler (AppleEvent *,AppleEvent *,SInt32);
OSErr     viewsFontChangeEventHandler   (AppleEvent *,AppleEvent *,SInt32);
OSErr     showPreferencesEventHandler   (AppleEvent *,AppleEvent *,SInt32);
OSErr     doHasGotRequiredParams        (AppleEvent *);
Boolean   doOpenFile                    (FSSpec *,SInt32,SInt32);
Boolean   doPrintFile                   (FSSpec *,SInt32,SInt32);
void      doPrepareToTerminate          (void);
WindowRef doNewWindow                   (void);
void      doMenuChoice                  (SInt32);
void      doErrorAlert                  (SInt16);
void      doDrawText                    (Str255);
void      doConcatPStrings              (Str255,Str255);

// ********************************************************************************
main

void  main(void)
{
  Rect          portRect;
  RGBColor      foreColor = {  0xFFFF,0xFFFF,0xFFFF } ;
  RGBColor      backColor = {  0x4444,0x4444,0x9999 } ;
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  EventRecord   eventStructure;

  // ..................................................................... do
preliminaries

  doPreliminaries();

  // ..................................................................... set up menu bar
and menus

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    doErrorAlert(eMenus);
  SetMenuBar(menubarHdl);
  DrawMenuBar();

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
      DisableMenuItem(menuRef,0);
    }

    gRunningOnX = true;

    EnableMenuCommand(NULL,kAEShowPreferences);
  }

  // .....................................................................
```

```
open a window

  if(!(gWindowRef = GetNewCWindow(rDisplayWindow,NULL,(WindowRef) -1)))
  {
    doErrorAlert(eCannotOpenWindow);
    ExitToShell();
  }

  SetPortWindowPort(gWindowRef);
  TextSize(10);
  TextFace(bold);
  RGBBackColor(&backColor);
  RGBForeColor(&foreColor);
  GetWindowPortBounds(gWindowRef,&portRect);
  EraseRect(&portRect);
  // ..................................................... install Apple event
handlers

  doInstallAEHandlers();

  // ................................................................... event
loop

  gDone = false;

  while(!gDone)
  {
    if(WaitNextEvent(everyEvent,&eventStructure,180,NULL))
      doEvents(&eventStructure);
  }
}

// ************************************************************************
doPreliminaries

void  doPreliminaries(void)
{
  MoreMasterPointers(64);
  InitCursor();
  FlushEvents(everyEvent,0);
}

// ************************************************************************
doInstallAEHandlers

void  doInstallAEHandlers(void)
{
  // ..................................................... required Apple
events

  doInstallAnAEHandler(kCoreEventClass,kAEOpenApplication,openAppEventHandler);
  doInstallAnAEHandler(kCoreEventClass,kAEReopenApplication,reopenAppEventHandler);
  doInstallAnAEHandler(kCoreEventClass,kAEOpenDocuments,openDocsEventHandler);
  doInstallAnAEHandler(kCoreEventClass,kAEPrintDocuments,printDocsEventHandler);
  doInstallAnAEHandler(kCoreEventClass,kAEQuitApplication,quitAppEventHandler);

  // ..................................................... Appearance Manager Apple
events

  doInstallAnAEHandler(kAppearanceEventClass,kAESystemFontChanged,
sysFontChangeEventHandler);
  doInstallAnAEHandler(kAppearanceEventClass,kAESmallSystemFontChanged,
                       smallSysFontChangeEventHandler);
  doInstallAnAEHandler(kAppearanceEventClass,kAEViewsFontChanged,
viewsFontChangeEventHandler);
```

```
  // ..................................................... Show Preferences Apple
event

  if(gRunningOnX)
    doInstallAnAEHandler(kCoreEventClass,kAEShowPreferences,showPreferencesEventHandler);
}

// *********************************************************************
doInstallAnAEHandler

void  doInstallAnAEHandler(AEEventClass eventClass,AEEventID eventID,void *theHandler)
{
  OSErr  osError;
  osError = AEInstallEventHandler(eventClass,eventID,
                                  NewAEEventHandlerUPP((AEEventHandlerProcPtr) theHandler),
                                  0L,false);
  if(osError != noErr)
    doErrorAlert(eInstallHandler);
}

// ***************************************************************************
doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
  WindowPartCode partCode;
  WindowRef      windowRef;
  SInt32         menuChoice;

  switch(eventStrucPtr->what)
  {
    case kHighLevelEvent:
      AEProcessAppleEvent(eventStrucPtr);
      break;

    case mouseDown:
      partCode = FindWindow(eventStrucPtr->where,&windowRef);
      switch(partCode)
      {
        case inMenuBar:
          menuChoice = MenuSelect(eventStrucPtr->where);
          doMenuChoice(menuChoice);
          break;

        case inDrag:
          DragWindow(windowRef,eventStrucPtr->where,NULL);
          break;

        case inContent:
          if(windowRef != FrontWindow())
            SelectWindow(windowRef);
          break;

        case inGoAway:
          DisposeWindow(windowRef);
          break;
      }
      break;

    case keyDown:
      if((eventStrucPtr->modifiers & cmdKey) != 0)
        doMenuChoice(MenuEvent(eventStrucPtr));
      break;

    case updateEvt:
      BeginUpdate((WindowRef)eventStrucPtr->message);
```

```
      EndUpdate((WindowRef)eventStrucPtr->message);
      break;
  }
}
// ************************************************************************
openAppEventHandler

OSErr  openAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefCon)
{
  OSErr     osError;
  WindowRef windowRef;

  gApplicationWasOpen = true;

  osError = doHasGotRequiredParams(appEvent);

  if(osError == noErr)
  {
    doDrawText("\pReceived an Apple event: OPEN APPLICATION.");
    doDrawText("\p    • Opening an untitled window in response.");

    windowRef = doNewWindow();
    SetWTitle(windowRef,"\puntitled");
  }

  return osError;
}

// ************************************************************************
reopenAppEventHandler

OSErr  reopenAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefCon)
{
  OSErr     osError;
  WindowRef windowRef;

  osError = doHasGotRequiredParams(appEvent);

  if(osError == noErr)
  {
    doDrawText("\pReceived an Apple event: RE-OPEN APPLICATION.");
    doDrawText("\p    • I will check whether I have any windows open.");

    windowRef = GetWindowList();

    if((windowRef = GetNextWindow(windowRef)) == NULL)
    {
      doDrawText("\p      No windows are open, so I will open a window.");

      windowRef = doNewWindow();
      SetWTitle(windowRef,"\puntitled 1");
    }
    else
      doDrawText("\p      A window is open, so I won't open another.");
  }

  return osError;
}
// ************************************************************************
openDocsEventHandler

OSErr  openDocsEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  AEDescList docList;
  OSErr      osError, ignoreErr;
  SInt32     numberOfItems, index;
```

```
  DescType   returnedType;
  FSSpec     fileSpec;
  AEKeyword  keyWord;
  Size       actualSize;
  Boolean    result;

  osError = AEGetParamDesc(appEvent,keyDirectObject,typeAEList,&docList);

  if(osError == noErr)
  {
    osError = doHasGotRequiredParams(appEvent);
    if(osError == noErr)
    {
      osError = AECountItems(&docList,&numberOfItems);
      if(osError == noErr)
      {
        for(index=1;index<=numberOfItems;index++)
        {
          osError = AEGetNthPtr(&docList,index,typeFSS,&keyWord,&returnedType,&fileSpec,
                                sizeof(fileSpec),&actualSize);
          if(osError == noErr)
          {
            if(!(result = doOpenFile(&fileSpec,index,numberOfItems)))
              doErrorAlert(eCannotOpenFile);
          }
          else
            doErrorAlert(eGetDescriptorRecord);
        }
      }
      else
        doErrorAlert(eCountDescripRecords);
    }
    else
      doErrorAlert(eMissedRequiredParam);

    ignoreErr = AEDisposeDesc(&docList);
  }
  else
    doErrorAlert(eGetRequiredParam);

  return osError;
}
// ************************************************************************
printDocsEventHandler

OSErr  printDocsEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  AEDescList docList;
  OSErr      osError, ignoreErr;
  SInt32     numberOfItems, index;
  DescType   returnedType;
  FSSpec     fileSpec;
  AEKeyword  keyWord;
  Size       actualSize;
  Boolean    result;

  osError = AEGetParamDesc(appEvent,keyDirectObject,typeAEList,&docList);

  if(osError == noErr)
  {
    osError = doHasGotRequiredParams(appEvent);
    if(osError == noErr)
    {
      osError = AECountItems(&docList,&numberOfItems);
      if(osError == noErr)
      {
```

```
        for(index=1;index<=numberOfItems;index++)
        {
          osError = AEGetNthPtr(&docList,index,typeFSS,&keyWord,&returnedType,&fileSpec,
                                sizeof(fileSpec),&actualSize);
          if(osError == noErr)
          {
            if(!(result = doPrintFile(&fileSpec,index,numberOfItems)))
              doErrorAlert(eCannotPrintFile);
          }
          else
            doErrorAlert(eGetDescriptorRecord);
        }
      }
      else
        doErrorAlert(eCountDescripRecords);
    }
    else
      doErrorAlert(eMissedRequiredParam);

    ignoreErr = AEDisposeDesc(&docList);
  }
  else
    doErrorAlert(eGetRequiredParam);

  return osError;
}
// *********************************************************************
quitAppEventHandler

OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  OSErr osError;

  osError = doHasGotRequiredParams(appEvent);

  if(osError == noErr)
    doPrepareToTerminate();

  return osError;
}

// *************************************************************
sysFontChangeEventHandler

OSErr  sysFontChangeEventHandler(AppleEvent *appEvent,AppleEvent *reply,
                                 SInt32 handlerRefcon)
{
  OSErr  osError;
  Rect   portRect;
  Str255 fontName, theString = "\p   Current large system font is: ";

  osError = doHasGotRequiredParams(appEvent);

  if(osError == noErr)
  {
    GetWindowPortBounds(gWindowRef,&portRect);
    EraseRect(&portRect);
    doDrawText("\pReceived an Apple event: LARGE SYSTEM FONT CHANGED.");
    GetThemeFont(kThemeSystemFont,smSystemScript,fontName,NULL,NULL);
    doConcatPStrings(theString,fontName);
    doDrawText(theString);
    // Action as required by application.
  }

  return osError;
}
```

```
// ************************************************************
smallSysFontChangeEventHandler

OSErr  smallSysFontChangeEventHandler(AppleEvent *appEvent,AppleEvent *reply,
                                      SInt32 handlerRefcon)
{
  OSErr  osError;
  Rect   portRect;
  Str255 fontName, theString = "\p   Current small system font is: ";

  osError = doHasGotRequiredParams(appEvent);

  if(osError == noErr)
  {
    GetWindowPortBounds(gWindowRef,&portRect);
    EraseRect(&portRect);
    doDrawText("\pReceived an Apple event: SMALL SYSTEM FONT CHANGED.");
    GetThemeFont(kThemeSmallSystemFont,smSystemScript,fontName,NULL,NULL);
    doConcatPStrings(theString,fontName);
    doDrawText(theString);
    // Action as required by application.
  }

  return osError;
}

// ************************************************************
viewsFontChangeEventHandler

OSErr  viewsFontChangeEventHandler(AppleEvent *appEvent,AppleEvent *reply,
                                   SInt32 handlerRefcon)
{
  OSErr  osError;
  Rect   portRect;
  Str255 fontName, fontSizeString, theString = "\p   Current views font is: ";
  SInt16 fontSize;

  osError = doHasGotRequiredParams(appEvent);
  if(osError == noErr)
  {
    GetWindowPortBounds(gWindowRef,&portRect);
    EraseRect(&portRect);
    doDrawText("\pReceived an Apple event: VIEWS FONT CHANGED.");
    GetThemeFont(kThemeViewsFont,smSystemScript,fontName,&fontSize,NULL);
    doConcatPStrings(theString,fontName);
    doConcatPStrings(theString,"\p ");
    NumToString((SInt32) fontSize,fontSizeString);
    doConcatPStrings(theString,fontSizeString);
    doConcatPStrings(theString,"\p point");
    doDrawText(theString);
    // Action as required by application.
  }

  return osError;
}

// ************************************************************
showPreferencesEventHandler

OSErr  showPreferencesEventHandler(AppleEvent *appEvent,AppleEvent *reply,
                                   SInt32 handlerRefcon)
{
  OSErr osError;
  Rect  portRect;
```

```
  osError = doHasGotRequiredParams(appEvent);
  if(osError == noErr)
  {
    GetWindowPortBounds(gWindowRef,&portRect);
    EraseRect(&portRect);
    doDrawText("\pReceived an Apple event: SHOW PREFERENCES.");
    doDrawText("\p    • I would present a Preferences... dialog now.");
  }

  return osError;
}

// **********************************************************************
doHasGotRequiredParams

OSErr  doHasGotRequiredParams(AppleEvent *appEvent)
{
  OSErr    osError;
  DescType returnedType;
  Size     actualSize;

  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,
NULL,0,
                              &actualSize);
  if(osError == errAEDescNotFound)
    osError = noErr;
  else if(osError == noErr)
    osError = errAEParamMissed;

  return osError;
}

// ****************************************************************************
doOpenFile

Boolean  doOpenFile(FSSpec *fileSpecPtr,SInt32 index,SInt32 numberOfItems)
{
  WindowRef windowRef;

  gApplicationWasOpen = true;

  if(index == 1)
    doDrawText("\pReceived an Apple event: OPEN DOCUMENTS.");

  if(numberOfItems == 1)
  {
    doDrawText("\p    • The file to open is: ");
    DrawString(fileSpecPtr->name);
    doDrawText("\p    • Opening titled window in response.");
  }
  else
  {
    if(index == 1)
    {
      doDrawText("\p    • The files to open are: ");
      DrawString(fileSpecPtr->name);
    }
    else
    {
      DrawString("\p and ");
      DrawString(fileSpecPtr->name);
      doDrawText("\p    • Opening titled windows in response.");
    }
  }

  if(windowRef = doNewWindow())
```

```
  {
    SetWTitle(windowRef,fileSpecPtr->name);
    return true;
  }
  else
    return false;
}

// ***************************************************************************
doPrintFile

Boolean  doPrintFile(FSSpec *fileSpecPtr,SInt32 index,SInt32 numberOfItems)
{
  WindowRef windowRef;
  UInt32    finalTicks;

  if(index == 1)
    doDrawText("\pReceived an Apple event: PRINT DOCUMENTS");

  if(numberOfItems == 1)
  {
    doDrawText("\p    • The file to print is: ");
    DrawString(fileSpecPtr->name);
    windowRef = doNewWindow();
    SetWTitle(windowRef,fileSpecPtr->name);
    Delay(60,&finalTicks);
    doDrawText("\p    • I would present the Print dialog first and then print");
    doDrawText("\p       the document when the user has made his settings.");
    Delay(60,&finalTicks);
    doDrawText("\p    • Assume that I am now printing the document.");
  }
  else
  {
    if(index == 1)
    {
      doDrawText("\p    • The first file to print is: ");
      DrawString(fileSpecPtr->name);
      doDrawText("\p       I would present the Print dialog for the first file");
      doDrawText("\p       only and use the user's settings to print both files.");
    }
    else
    {
      doDrawText("\p    • The second file to print is: ");
      DrawString(fileSpecPtr->name);
      doDrawText("\p       I am using the Print dialog settings used for the");
      doDrawText("\p       first file.");
    }
    windowRef = doNewWindow();
    SetWTitle(windowRef,fileSpecPtr->name);
    doDrawText("\p    • Assume that I am now printing the document.");
  }

  if(numberOfItems == index)
  {
    if(!gApplicationWasOpen)
    {
      doDrawText("\p       Since the application was not already open, I expect to");
      doDrawText("\p       receive a QUIT APPLICATION event when I have finished.");
    }
    else
    {
      doDrawText("\p       Since the application was already open, I do NOT expect");
      doDrawText("\p       to receive a QUIT APPLICATION event when I have finished.");
    }

    Delay(180,&finalTicks);
```

```
    doDrawText("\p    • Finished print job.");
  }
  else
    Delay(180, &finalTicks);

  DisposeWindow(windowRef);
  return true;
}

// ************************************************************************
doPrepareToTerminate

void  doPrepareToTerminate(void)
{
  UInt32 finalTicks;

  doDrawText("\pReceived an Apple event: QUIT APPLICATION");

  if(gApplicationWasOpen)
  {
    doDrawText("\p    • I would now ask the user to save any unsaved files before");
    doDrawText("\p       terminating myself in response to the event.");
    doDrawText("\p    • Click the mouse when ready to terminate.");
    while(!Button()) ;
  }
  else
  {
    doDrawText("\p    • Terminating myself in response");
    Delay(240,&finalTicks);
  }

  // If the user did not click the Cancel button in a Save dialog:

  gDone = true;
}
// ************************************************************************
doNewWindow

WindowRef  doNewWindow(void)
{
  WindowRef windowRef;

  if(!(windowRef = GetNewCWindow(rDocWindow,NULL,(WindowRef) -1)))
    doErrorAlert(eCannotOpenWindow);

  return windowRef;
}

// ************************************************************************
doMenuChoice

void  doMenuChoice(SInt32 menuChoice)
{
  MenuID        menuID;
  MenuItemIndex menuItem;

  menuID   = HiWord(menuChoice);
  menuItem = LoWord(menuChoice);

  if(menuID == 0)
    return;

  switch(menuID)
  {
    case mFile:
      if(menuItem  == iQuit)
```

```
        gDone = true;
      break;
  }

  HiliteMenu(0);
}

// ********************************************************************************
doErrorAlert

void  doErrorAlert(SInt16 errorType)
{
  Str255 errorString;
  SInt16 itemHit;

  GetIndString(errorString,rErrorStrings,errorType);

  if(errorType < 7)
    StandardAlert(kAlertCautionAlert,errorString,NULL,NULL,&itemHit);
  else
  {
    StandardAlert(kAlertStopAlert,errorString,NULL,NULL,&itemHit);
    ExitToShell();
  }
}
// ********************************************************************************
doDrawText

void  doDrawText(Str255 eventString)
{
  RgnHandle tempRegion;
  SInt16    a;
  Rect      portRect;
  UInt32    finalTicks;

  tempRegion = NewRgn();
  GetWindowPortBounds(gWindowRef,&portRect);

  for(a=0;a<15;a++)
  {
    ScrollRect(&portRect,0,-1,tempRegion);
    QDFlushPortBuffer(GetWindowPort(gWindowRef),NULL);

    Delay(4,&finalTicks);
  }

  DisposeRgn(tempRegion);

  MoveTo(8,160);
  DrawString(eventString);
  QDFlushPortBuffer(GetWindowPort(gWindowRef),NULL);
}

// ********************************************************************************
doConcatPStrings

void  doConcatPStrings(Str255 targetString,Str255 appendString)
{
  SInt16 appendLength;

  appendLength = MIN(appendString[0],255 - targetString[0]);

  if(appendLength > 0)
  {
    BlockMoveData(appendString+1,targetString+targetString[0]+1,(SInt32) appendLength);
    targetString[0] += appendLength;
```

```
  }
}

//
*****************************************************************************************
```

Demonstration Program AppleEvents Comments

The demonstration requires that the user open the windows containing the AppleEvents application icon and the two document file icons AppleEvents Document A and AppleEvents Document B.

Using all of the methods available in the Finder (that is, double clicking the icons, dragging document icons to the application icon, selecting the icons and choosing Open and (on Mac OS 8/9 only) Print from the Finder's File menu, choosing the application from the Apple menu), the user should launch the application, open the simulated documents and (on Mac OS 8/9 only) "print" the documents, noting the descriptive text printed in the non-document window in response to the receipt of the resulting Apple events.

When the application is running, the user should double-click the application icon, choose the application from the Mac OS 8/9 Apple menu, or click the application icon and choose Open from the Finder's File menu, noting the receipt of the Re-open Application event. This should be done when one or more document windows are open and when no document windows are open.

The user should also choose Restart or Shut Down while the application is running, also noting the displayed text resulting from receipt of the Quit Application event. (On Mac OS X, the Quit Application event is also received when Quit is chosen from the Application menu.)

On Mac OS 8/9, opening and printing should be attempted (from the Finder's File menu) when the application is already running and when the application is not running.

On Mac OS X, the user should choose Show Preferences… from the Application menu.

Note that, in this demonstration, it is possible to "open" each document more than once, that is, it is possible to have several "Document A" and/or "Document B" windows open at once. (A real application would permit only one "Document A" and/or one "Document B" window to be open at any one time.) The reason for this is that, in this demonstration, files are not actually opened; accordingly, it is not possible to check whether the relevant file is already open.

With regard to the Appearance Manager Apple events, the user should make changes to the system font, small system font, and views font in the Fonts tab of the Appearance control panel, noting the descriptive text that appears in the non-document window.

Although not related to the required Apple events aspects of the program, the following aspects of the demonstration may also be investigated:

- On Mac OS 8/9, the customized finder icon help override help balloon for the application icon. (The 'hfdr' resource refers.)
- The version information for the application in the Finder's Get Info (Mac OS 8/9) and Show Info (Mac OS X) window and in the window containing the AppleEvents application when list view is selected.

'plst' Resource

The following is the information property list in the application's 'plst' resource:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist SYSTEM "file://localhost/System/Library/DTDs/PropertyList.dtd">
<plist version="0.9">
```

```
<dict>
  <key>CFBundleInfoDictionaryVersion</key>
  <string>6.0</string>
  <key>CFBundleName</key>
  <string>AppleEvents</string>
  <key>CFBundlePackageType</key>
  <string>APPL</string>
  <key>CFBundleSignature</key>
  <string>KJBB</string>
  <key>CFBundleIconFile</key>
  <string>128</string>
  <key>CFBundleIdentifier</key>
  <string>com.Windmill Software.AppleEvents</string>
  <key>LSPrefersCarbon</key>
  <string>Yes</string>
  <key>CFBundleLongVersionString</key>
  <string>Version 1.0.0 June 2001</string>
  <key>CFBundleShortVersionString</key>
  <string>1.0.0</string>
  <key>CFBundleDevelopmentRegion</key>
  <string>English</string>
  <key>NSAppleScriptEnabled</key>
  <string>No</string>
  <key>CFBundleDocumentTypes</key>
  <array>
    <dict>
      <key>CFBundleTypeIconFile</key>
      <string>129</string>
      <key>CFBundleTypeName</key>
      <string>AppleEvents Document</string>
      <key>CFBundleTypeOSTypes</key>
      <array>
        <string>TEXT</string>
      </array>
      <key>CFBundleTypeRole</key>
      <string>Viewer</string>
    </dict>
  </array>
</dict>
</plist>
```

Global Variables

`gRunningOnX` will be set to true if the program is running on Mac OS X.

`gApplicationWasOpen` will be used to control the manner of program termination when a Quit Application event is received, depending on whether the event followed a Print Documents event (on Mac OS 8/9) or resulted from the user choosing Restart of Shut Down from the Finder's Special menu.

main

In the menus setting up block, if the program is running on Mac OS X, the Preferences item in the Mac OS X Application menu is enabled.

The call to `doInstallAEHandlers` installs the Apple event handlers.

doInstallAEHandlers and doInstallAnAEHandler

`doInstallAEHandlers`, supported by `doInstallAnAEHandler`, installs the handlers for the required Apple events, the Appearance Manager Apple events, and (if the program is running on Mac OS X) the Show Preferences Apple event in the application's Apple event dispatch table.

In `doInstallAnAEHandler`, false is passed in `AEInstallEventHandler`'s `isSysHandler` parameter. False causes the handler to be installed in the application's Apple event dispatch table. True causes handlers to be installed in the system's Apple event dispatch table. (The system Apple event dispatch table is a table in the system heap containing handlers that are available to all applications and processes running on the same computer.)

doEvents

The `kHighLevelEvent` case accommodates the receipt of a high-level event, in which case `AEProcessAppleEvent` is called. (`AEProcessAppleEvent` looks in the application's Apple event dispatch table for a match to the event class and event ID contained in, respectively, the event structure's message and where fields, and calls the appropriate handler.)

doOpenAppEvent

`doOpenAppEvent` is the handler for the Open Application event.

At the first line, the global variable `gApplicationWasOpen`, which controls the manner of program termination when a Quit Application event is received, is set to true. (This line is required for demonstration program purposes only.)

The function `doHasGotRequiredParams` is then called to check whether the Apple event contains any required parameters. If so, the handler returns an error because, by definition, the Open Application event should not contain any required parameters.

If `noErr` is returned by `doHasGotRequiredParams`, the handler does what the user expects the application to do when it is opened, that is, it opens an untitled document window (the call to `doNewWindow` and the subsequent call to `SetWTitle`). The handler then returns `noErr`.

If `errAEParamMissed` is returned by `doHasGotRequiredParams`, this is returned by the handler.

The calls to `doDrawText` simply print some text in the text window for demonstration program purposes.

doReopenAppEvent

`doRepenAppEvent` is the handler for the Re-open Documents event.

At the first line, the function `doHasGotRequiredParams` is called to check whether the Apple event contains any required parameters. If so, the handler returns an error because, by definition, the Re-open Application event should not contain any required parameters.

If `noErr` is returned by `doHasGotRequiredParams`, and if there are currently no open windows, the handler opens an untitled document window and returns `noErr`.

If `errAEParamMissed` is returned by `doHasGotRequiredParams`, this is returned by the handler.

The calls to doDrawText simply print some text in the text window for demonstration program purposes.

doOpenDocsEvent

doOpenDocsEvent is the handler for the Open Documents event.

At the first line, AEGetParamDesc is called to get the direct parameter (specified in the keyDirectObject keyword) out of the Apple event. The constant typeAEList specifies the descriptor type as a list of descriptor structures. The descriptor list is received by the docList variable.

Before proceeding further, the handler checks that it has received all the required parameters by calling the function doHasGotRequiredParams.

Having retrieved the descriptor list from the Apple event, the handler calls AECountItems to count the number of descriptors in the list.

Using the returned number as an index, AEGetNthPtr is called to get the data of each descriptor structure in the list. In the AEGetNthPtr call, the parameter typeFSS specifies the desired type of the resulting data, causing the Apple Event Manager to coerce the data in the descriptor structure to a file system specification structure. Note also that keyWord receives the keyword of the specified descriptor structure, returnedType receives the descriptor type, fileSpec receives a pointer to the file system specification structure, sizeof(fileSpec) establishes the length, in bytes, of the data returned, and actualSize receives the actual length, in bytes, of the data for the descriptor structure.

After extracting the file system specification structure describing the document to open, the handler calls the function for opening files (doOpenFile). (In a real application, that function would typically be the same as that invoked when the user chooses Open from the application's File menu.)

If the call to AEGetNthPtr does not return noErr, the error handling function (doErrorAlert) is called. (AEGetNthPtr will return an error code if there was insufficient room in the heap, the data could not be coerced, the descriptor structure was not found, the descriptor was of the wrong type, or the descriptor structure was not a valid descriptor structure.)

If the call to doHasGotRequiredParams does not return noErr, the error handling function (doErrorAlert) is called. (doHasGotRequiredParams returns noErr only if you got all the required parameters.)

Since the handler has no further requirement for the data in the descriptor list, AEDisposeDesc is called to dispose of the descriptor list.

If the call to AEGetParamDesc does not return noErr the error handling function (doErrorAlert) is called. (AEGetParamDesc will return an error code for much the same reasons as will AEGetNthPtr.)

doPrintDocsEvent

doPrintDocsEvent is the handler for the Print Documents event.

The code is identical to that for the Open Documents event handler doOpenDocs except that the function for printing files (doPrintFile) is called rather than the function for simply opening files (doOpenFile).

doQuitAppEvent

`doQuitAppEvent` is the handler for the Quit Application event.

After checking that it has received all the required parameters by calling the function `doHasGotRequiredParams`, the handler calls the function `doPrepareToTerminate`.

doSysFontChangeEvent, doSmallSysFontChange, and doViewsFontChangeEvent

`doSysFontChangeEvent`, `doSmallSysFontChange`, and `doViewsFontChangeEvent` are the handlers for the appearance Apple events.

The function `doHasGotRequiredParams` is called to check whether the Apple event contains any required parameters. If so, the handler returns an error because, by definition, these events should not contain any required parameters.

The handlers then draw some advisory text in the non-document window indicating that the event has been received, call the Appearance Manager function `GetThemeFont` to obtain information about the relevant font (large system, small system, or views), and draw the font name (and, in the case of the views font, the font size).

showPreferencesEventHandler

`showPreferencesEventHandler` is the handler for the Show Preferences Apple event. It simply draws some advisory text in the window to prove that the event was received.

doHasGotRequiredParams

`doHasGotRequiredParams` is the function called by `doOpenAppEvent`, `doReopenAppEvent`, `doSysFontChangeEvent`, `doSmallSysFontChange`, and `doViewsFontChangeEvent` to confirm that the event passed to it contains no required parameters, and by the other required Apple event handlers to check that they have received all the required parameters.

The first parameter in the call to `AEGetAttributePtr` is a pointer to the Apple event in question. The second parameter is the Apple event keyword; in this case the constant `keyMissedKeywordAttr` is specified, meaning the first required parameter remaining in the event. The third parameter specifies the descriptor type; in this case the constant `typeWildCard` is specified, meaning any descriptor type. The fourth parameter receives the descriptor type of the returned data. The fifth parameter is a pointer to the data buffer, which stores the returned data. The sixth parameter is the maximum length of the data buffer to be returned. Since we do not need the data itself, these parameters are set to `NULL` and `0` respectively. The last parameter receives the actual length, in bytes, of the data buffer for the attribute.

`AEGetAttributePtr` returns the result code `errAEDescNotFound` if the specified descriptor type (`typeWildCard`, that is, any descriptor type) is not found, meaning that the handler extracted all the required parameters. In this event, `doHasGotRequiredParams` returns noErr.

If `AEGetAttributePtr` returns `noErr`, the handler has not extracted all of the required parameters, in which case, the handler should return `errAEParamMissed` and not handle the event. Accordingly, `errAEParamMissed` is returned to the handler (and, in turn, by the handler) if `noErr` is returned by `AEGetAttributePtr`.

doOpenFile

`doOpenFile` takes the file system specification structure and opens a window with the filename contained in that structure repeated in the window's title bar (the calls to `doNewWindow` and `SetWTitle`). The rest of the `doOpenFile` code simply draws explanatory text in the text window.

In a real application, this is the function that would open files as a result of, firstly, the receipt of the Open Documents event and, secondly, the user choosing Open from the application's File menu and then choosing a file or files from the resulting Navigation Services Open dialog.

doPrintFile

`doPrintFile` is applicable to Mac OS 8/9 only. It is the function which, in a real application, would take the file system specification structure passed to it from the Print Documents event handler, extract the filename and control the printing of that file. In this demonstration, most of the `doPrintFile` code is related to drawing explanatory text in the text window.

If your application can interact with the user, it should open windows for the documents, display a Print dialog for the first document, and use the settings entered by the user for the first document to print all documents.

Note that if your application was not running when the user selected a document icon and chose Print from the Finder's File menu, the Finder will send a Quit Application event following the print operation.

doPrepareToTerminate

`doPrepareToTerminate` is the function called by the Quit Application event handler. In this demonstration, `gDone` will be set to true, and the program will thus terminate immediately, if the Quit Application event resulted from the user initiating a print operation from the Mac OS 8/9 Finder when the application was not running.

If the application was running (`gApplicationWasOpen` contains true) and the Quit Application event thus arose from the user selecting Restart or Shut Down from the Finder's File menu, the demonstration waits for a button click before setting `gDone` to true. (In a real application, and where appropriate, this area of the code would invoke alerts to ascertain whether the user wished to save changed documents before closing down.)

Note that, when your application is ready to quit, it must call `ExitToShell` from the main event loop, not from the handlers for the Quit Application event. Your application should quit only after the handler returns `noErr` as its function result.

doNewWindow

`doNewWindow` opens document windows in response to calls from the Open Application and Open Documents event handlers.

Chapter 11. QuickDraw Preliminaries

IN THIS CHAPTER

- QuickDraw and Imaging
- RGB Colors and Pixels
- Color and the Video Device
- Graphics Port
- Translation of RGB Colors to Pixel Values
- Graphics Devices and GDevice Structures
- Other Graphics Managers
- Coping with Multiple Monitors
- Relevant QuickDraw Constants, Data Types, and Functions
- Demonstration Program PreQuickDraw Listing
- Demonstration Program PreQuickDraw Comments

QuickDraw and Imaging

QuickDraw is a collection of system software routines that your application uses to perform **imaging** operations, that is, the construction and display of graphical information for display on output devices such as screens and printers.

This chapter serves as a prelude to Chapter 12, and introduces certain matters that need to be discussed before the matter of actually drawing with QuickDraw is addressed. These matters include RGB colors, color and the video device, the graphics port, translation of RGB values, and graphics devices.

RGB Colors and Pixels

In QuickDraw, colors are specified as **RGB colors** using an `RGBColor` structure:

```
struct RGBColor
{
  unsigned short red;    // Magnitude of red component.
  unsigned short green;  // Magnitude of green component.
  unsigned short blue;   // Magnitude of blue component.
} ;
typedef struct RGBColor RGBColor;
```

Note that an RGB color is defined by three components (red, green and blue). When the `red`, `green` and `blue` fields of the `RGBColor` structure are assigned the maximum possible value (`0xFFFF`), the resulting color is white. When these fields are assigned the minimum value (`0x0000`), the resulting color is black.

A **pixel** (picture element) is the smallest dot that QuickDraw can draw. Each color pixel represents up to 48 bits in memory.

Color and the Video Device

QuickDraw supports a variety of screens of differing sizes and color capabilities, and is thus device-independent. Accordingly, you do not have to concern yourself with the capabilities of individual screens. For example, when your application uses an `RGBColor` structure to specify a color by its red, green and blue components, with each component defined in a 16-bit integer, QuickDraw compares the resulting 48-bit value with the colors actually available on a video device (such as a plug-in video card or a built-in video interface) at execution time and then chooses the closest match. What the user finally sees depends on the characteristics of the actual video device and screen.

The video device that controls a screen may have either:

- **Indexed colors,** which support pixels of 1-bit, 2-bit, 4-bit, or 8-bit pixel depths. The indexed color system was introduced with the Macintosh II, that is, at a time when memory was scarce and moving megabyte images around was quite impractical.
- **Direct colors,** which support pixels of 16-bit and 32-bit depths. Most video devices in the current day are direct color devices. (However, as will be seen, there are circumstances in which a direct color device will act like an indexed color device.)

> **Note**
>
> Pixel depth means the number of bits assigned to each pixel, and thus determines the maximum number of colors that can be displayed at the one time. A 4-bit pixel depth, for example, means that an individual pixel can be displayed in any one of 16 separate colors. An 8-bit pixel depth means that an individual pixel can be displayed in any one of 256 separate colors.

QuickDraw automatically determines which method is used by the video device and matches your requested 48-bit color with the closest available color.

Indexed Color Devices

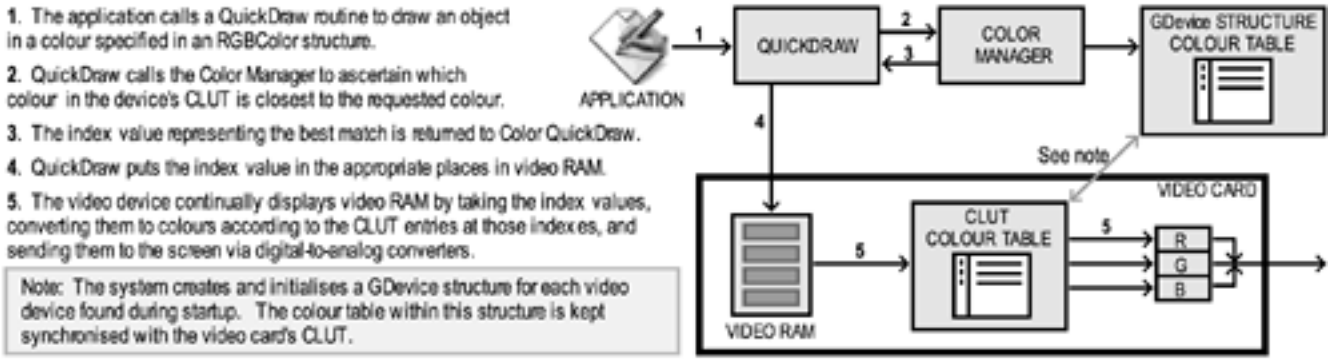
Video devices using indexed colors support a maximum of 256 colors at any one time, that is, with indexed color, the maximum value of a pixel is limited to a single byte, with each pixel's byte specifying one of 256 different values.

Video devices implementing indexed color contain a data structure called a **color lookup table** (**CLUT**), which contains entries for all possible color values. Most indexed video devices use a

variable CLUT, which allows your application to load the CLUT with different sets of colors depending on the image being displayed.

When your application uses a 48-bit `RGBColor` structure to specify a color, the Color Manager compares the CLUT entries on the video device with the specified `RGBColor` color, determines which color in the CLUT is closest, and passes QuickDraw the index to this color. This is the color that QuickDraw draws with. Fig 1 illustrates this process.

Figure 1. INDEXED COLOUR SYSTEM


1. The application calls a QuickDraw routine to draw an object in a colour specified in an RGBColor structure.
2. QuickDraw calls the Color Manager to ascertain which colour in the device's CLUT is closest to the requested colour.
3. The index value representing the best match is returned to Color QuickDraw.
4. QuickDraw puts the index value in the appropriate places in video RAM.
5. The video device continually displays video RAM by taking the index values, converting them to colours according to the CLUT entries at those indexes, and sending them to the screen via digital-to-analog converters.
Note: The system creates and initialises a GDevice structure for each video device found during startup. The colour table within this structure is kept synchronised with the video card's CLUT.
APPLICATION
QUICKDRAW
COLOR MANAGER
GDevice STRUCTURE
COLOUR TABLE
See note
VIDEO CARD
CLUT
COLOUR TABLE
VIDEO RAM
R
G
B

Direct Color Devices

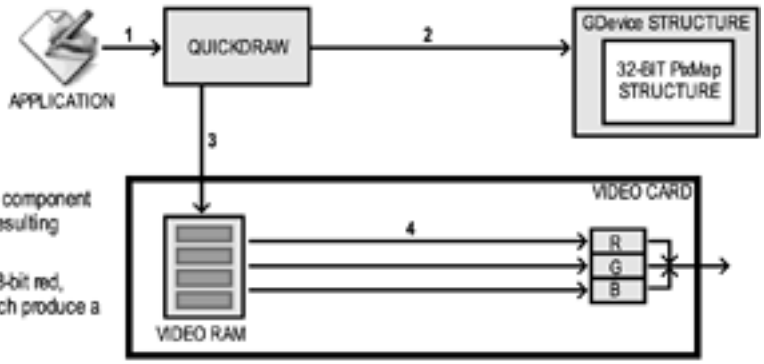
Video devices which implement direct color eliminate the competition for limited color lookup table spaces and remove the need for color table matching. By using direct color, video devices can support thousands or millions of colors.

When you specify a color using a 48-bit `RGBColor` structure on a direct color system, QuickDraw truncates the least significant bits of its red, green and blue components to either 16 bits (five bits each for red, green and blue, with one bit unused) or 32 bits (eight bits for red, green and blue, with eight bits unused). (See "Translation of RGB Colors to Pixel Values.") Using 16 bits, direct video devices can display 32,768 different colors. Using 32 bits, the device can display 16,777,215 different colors

Fig 2 illustrates the direct color system.

Figure 2. DIRECT COLOUR SYSTEM

1. The application calls a QuickDraw routine to draw an object in a colour specified in an RGBColor structure.
2. QuickDraw knows from the GDevice structure that the screen is controlled by a direct device in which pixels are, say, 32 bits deep, which means that eight bits are used for each of the red, green, and blue components of the requested colour.
3. Accordingly, QuickDraw passes the high eight bits from each 16-bit component of the 48-bit RGBColor structure to the video device, which stores the resulting 24-bit value in video RAM for the object.
4. The video device continually displayes video RAM by sending the 8-bit red, green, and blue values for the colour to digital-to-analog converters which produce a signal for the screen



Direct color not only removes much of the complexity of the CLUT mechanism for video device developers, but also allows the display of thousands or millions of colors simultaneously, resulting in near-photographic resolution.

Direct Devices Operating Like Indexed Devices

Note that, when a user sets a direct color device to use 256 colors (or less) as either a grayscale or color device, the direct device creates a CLUT and operates like an indexed device.

Graphics Port

A graphics port defines a complete drawing environment. Among other things, a graphics port

- Contains a handle to a **pixel map** that, in turn, contains a pointer to the area of memory in which your drawing operations take place.
- Contains a metaphorical graphics **pen** with which to perform drawing operations. (You can set this pen to different sizes, patterns and colors.)
- Holds information about text, which is styled and sized according to information in the graphics port.

> **Note**
>
> The term "graphics port" originally pertained to the one-bit graphics port used by the early black-and-white Macintoshes. The color graphics port was introduced when color came to the Macintosh with the Macintosh II. In the Carbon era, the black-and-white graphics port is irrelevant. Accordingly, where the term "graphics port" is used in this book, a color graphics port should be assumed unless otherwise stated.

The information in a graphics port is maintained by QuickDraw.

The graphics port is an opaque data structure. The data types `CGrafPtr` and `GrafPtr` are defined as pointers to such objects:

```
typedef struct OpaqueGrafPtr* GrafPtr;
typedef GrafPtr CGrafPtr;
```

Accessor Functions

Accessor functions are provided to access the information in color graphic port objects. The main accessor functions are as follows:

Accessor Function	*Description*
`GetPortPixMap`	Get a handle to the graphics port's pixel map

`GetPortBounds`
`SetPortBounds`

Get and set the graphics port rectangle.

Your application's drawing operations take place inside the port rectangle (which, for a window's graphics port is also called the **content region**).

The port rectangle uses a local coordinate system in which the upper-left corner of the port rectangle has a vertical coordinate of 0 and a horizontal coordinate of 0.

`GetPortVisRegion`
`SetPortVisRegion`

Get and set the visible region.

The **visible region** (which, by default, is equivalent to the port rectangle) is the region of the graphics port that is actually visible on screen (see Fig 3).

`GetPortClipRegion`
`SetPortClipRegion`
`SetClip`

Get and set the clipping region.

The **clipping region** is an arbitrary region used to limit drawing to any region within the port rectangle. The default clipping region is set arbitrarily large; however, your application can change this. In Fig 3, for example, `SetPortClipRegion` (or `ClipRect`) has been used to change Window B's clipping region so as to prevent the scroll bar areas being overdrawn.

`GetPortForeColor`
`RGBForeColor`

Get and set the foreground color.

These functions get and set an `RGBColor` structure that contains the *requested* **foreground color.** By default, the foreground color is black.

`GetPortBackColor`
`RGBBackColor`

Get and set the background color.

These functions get and set an `RGBColor` structure that contains the *requested* **background color.** By default, the background color is white.

`GetPortBackPixPat`
`SetPortBackPixPat`
`BackPixPat`
`BackPat`
	Get and set the background pixel pattern.
	
	These functions get and set a handle to a `PixPat` structure that describes the background **pixel pattern.** Various QuickDraw functions use this pattern for filling scrolled or erased areas.

`GetPortPenPixPat`
`SetPortPenPixPat`
`PenPixPat`
`PenPat`
	Get and set the pen pixel pattern.
	
	These functions get and set a handle to a `PixPat` structure that describes the pixel pattern used by the graphics pen for drawing lines and framed shapes, and for painting shapes.

`GetPortFillPixPat`	Get the fill pixel pattern. This function gets a handle to a `PixPat` structure that describes the pixel pattern used when you call QuickDraw shape filling functions.
`GetPortPenLocation` `MoveTo`	Get and set the pen location. The pen location is the point where QuickDraw will begin drawing the next line, shape, or character. It can be anywhere on the coordinate plane.

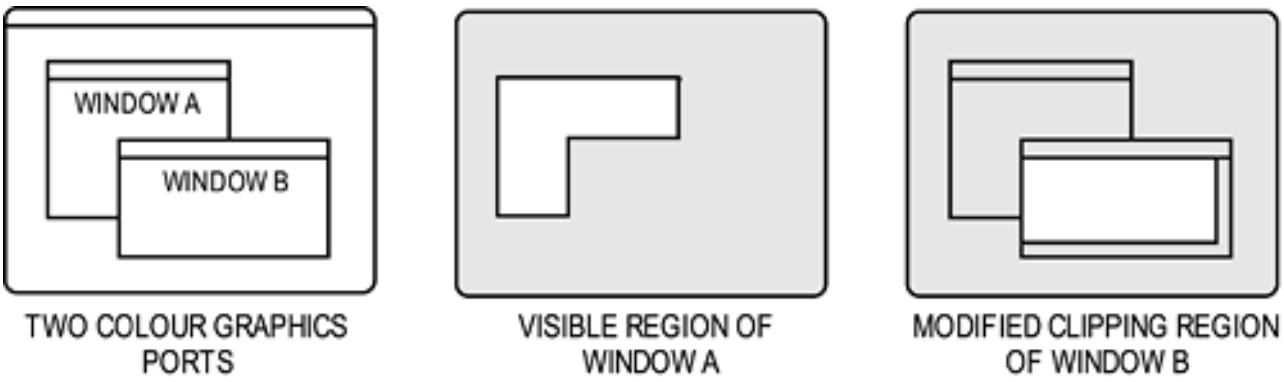
`GetPortPenSize`
`SetPortPenSize`
`PenSize`
	Get and set the pen size.
	
	Pen size is the vertical height and horizontal width of thegraphics pen. The default size is a 1-by-1 pixel square. If either the pen width or height is `0`, the pen does not draw.

`GetPortPenMode`
`SetPortPenMode`
`PenMode`
	Gets and sets the pen transfer mode.
	
	The pen **transfer mode** is a Boolean or arithmetic operation that determines how QuickDraw transfers the pen pattern to the pixel map during drawing operations. (See Chapter 12.)

`HidePen/ShowPen` `GetPortPenVisibility`	Gets and sets pen visibility. The pen's **visibility** means whether it draws on the screen.

`GetPortTextFont` `TextFont`	Get and set the font number for text. These functions get and set a **font family ID,** that is, a number that identifies the font to be used in the graphics port.
`GetPortTextSize` `TextSize`	Get and set the text size. The text size is expressed in pixels, and is used by the Font Manager to provide the bitmaps for text drawing.
`GetPortTextFace` `TextFace`	Get and set the text style. The **style** of the text means, for example, bold, italic, and/or underlined.
`GetPortTextMode` `TextMode`	Get and set the text mode. The text mode is the transfer mode for text drawing, which functions much like the transfer mode specified in the `pnMode` field.
`HiliteColor`	Get the highlight color. (The highlight color is copied to the graphics port from the low memory global `HiliteRGB`.)

Figure 3. VISIBLE REGION AND CLIPPING REGION



You can open many graphics ports at the same time. Each has its own local coordinate system, drawing pattern, background pattern, pen size and location, foreground color, background color, pixel map, and so on. You can instantly switch from one graphics port to another using the functions `SetPort`, `SetPortDialogPort`, and `SetPortWindowPort`.

When you use Window Manager and Dialog Manager functions to create windows, dialogs, and alerts, those managers automatically create graphics ports for you.

Pixel Maps

QuickDraw draws in a **pixel map.** The graphics port object contains a handle to a pixel map, which is a data structure of type `PixMap`. A `PixMap` structure contains a pointer to a **pixel image,** as well as information on the image's storage format, depth, resolution, and color usage. The `PixMap` structure is as follows:

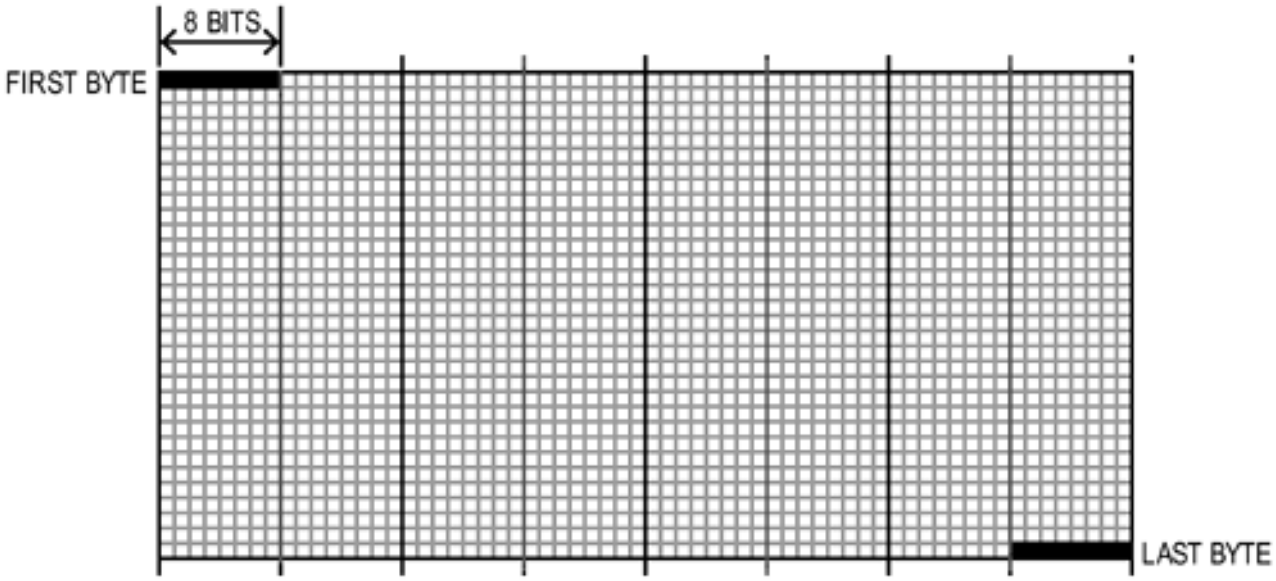
```
struct PixMap
{
  Ptr        baseAddr;    // Pointer to image data.
  short      rowBytes;    // Flags, and bytes in a row.
  Rect       bounds;      // Boundary rectangle.
  short      pmVersion;   // Pixel Map version number.
  short      packType;    // Packing format.
  long       packSize;    // Size of data in packed state.
  Fixed      hRes;        // Horizontal resolution in dots per inch.
  Fixed      vRes;        // Vertical resolution in dots per inch.
  short      pixelType;   // Format of pixel image.
  short      pixelSize;   // Physical bits per pixel.
  short      cmpCount;    // Number of components in each pixel.
  short      cmpSize;     // Number of bits in each component.
  long       planeBytes;  // Offset to next plane.
  CTabHandle pmTable;     // Handle to a color table for this image.
  long       pmReserved;  // (Reserved.)
};
typedef struct PixMap PixMap,*PixMapPtr,**PixMapHandle;
```

Field Descriptions

`baseAddr` In the case of an onscreen pixel image, a pointer to the first byte of the image data. Note that there can be several pixel maps pointing to the same pixel image, each imposing its own coordinate system on it.

A pixel image is analogous to the **bit image.** A bit image is a collection of bits in memory that form a grid. Fig 4 illustrates a bit image, which can be visualized as a matrix of rows and columns of bits with each row containing the same number of bytes. Each bit corresponds to one screen pixel. If a bit's value is 0, its screen pixel is white; if the bit's value is 1, the screen pixel is black. A pixel image is essentially the same as a bit image, except that a number of bits, not just one bit, are assigned to each pixel. The number of bits per pixel in a pixel image is called the pixel depth.

Figure 4. A BIT IMAGE

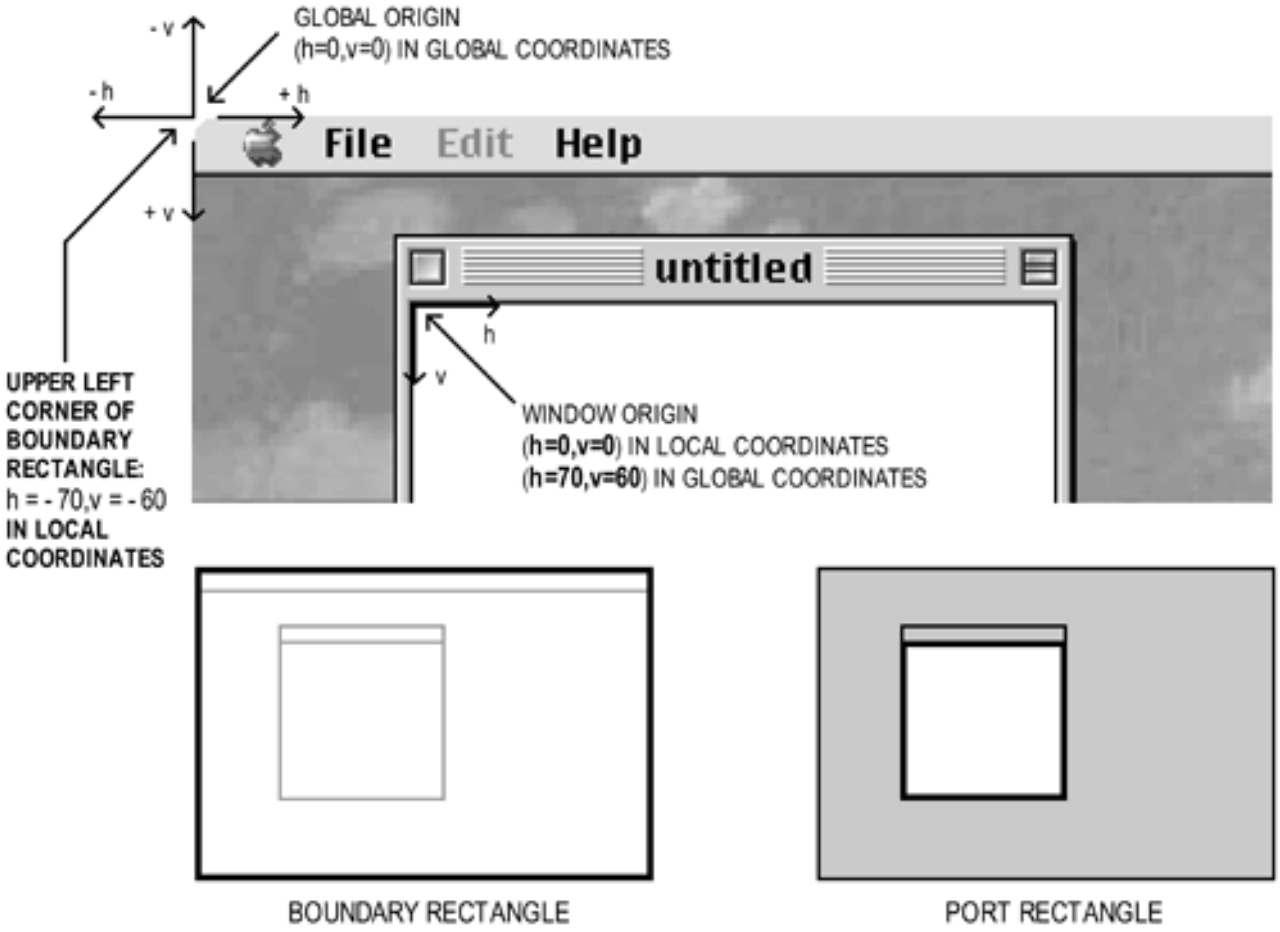


`rowBytes` The offset in bytes from one row of the image to the next.

`bounds` On Mac OS 8/9,the boundary rectangle defines the area of the pixel image into which QuickDraw can draw and provides the link between the local coordinate system of a graphics port and QuickDraw's global coordinate system. All drawing in a graphics port occurs in the intersection of the boundary rectangle and the port rectangle (and, within that intersection, all drawing is cropped to the graphics port's visible region and its clipping region).

As shown in Fig 5, on Mac OS 8/9, QuickDraw assigns the entire screen as the boundary rectangle. The boundary rectangle shares the same local coordinate system as the port rectangle of the window.

Figure 5. LOCAL AND GLOBAL COORDINATE SYSTEMS, THE BOUNDARY RECTANGLE AND THE PORT RECTANGLE-MAC OS 8/9



You should not, incidentally, use the `bounds` field to determine the size of the screen; instead, use the `gdRect` field of the `GDevice` structure.

On Mac OS X, the `bounds` field contains the bounds of the Core Graphics window that backs the Carbon window, and different mechanisms are employed to determine where the window's pixel map should be drawn.

`pmVersion` The QuickDraw version number that created this `PixMap` structure.

`packType` The packing algorithm used to compress the image data.

`packSize` The size of the packed image.

`hRes` The horizontal resolution of the pixel image in pixels per inch, abbreviated as dpi (dots per inch). By default, the value here is 0x00480000 (for 72 dpi), but QuickDraw supports `PixMap` structures of other resolutions. For example, `PixMap` structures for scanners can have dpi resolutions of 150, 200, 300, or greater.

`vRes` The vertical resolution. (See `hRes`.)

`pixelType` The storage format. 0 indicates indexed pixels. 16 (`RGBDirect`) indicates direct pixels.

`pixelSize` The number of bits used to represent a pixel.

`cmpCount` The number of components used to represent a color for a pixel. For indexed pixels, this field contains 1. For direct pixels this field contains the value 3.

`cmpSize` The size of each color component. For indexed devices, this is the same value as that in the `pixelSize` field. For direct devices, each of the three color components can be either 5 bits for a 16-bit pixel (one of these 16 bits is unused), or 8 bits for a 32-bit pixel (8 of these 32 bits are unused). (See "Translation of RGB Colors to Pixel Values.")

`planeBytes` Multiple-plane images are not supported, so the value of this field is always 0.

`pmTable` A handle to the `ColorTable` structure. `ColorTable` structures define the colors available for pixel images on indexed devices. Pixel images on direct devices do not need a color table because the colors are stored right in the pixel values. In the case of direct devices, `pmTable` points to a dummy color table.

Functions

Carbon introduced the following functions relating to pixel maps:

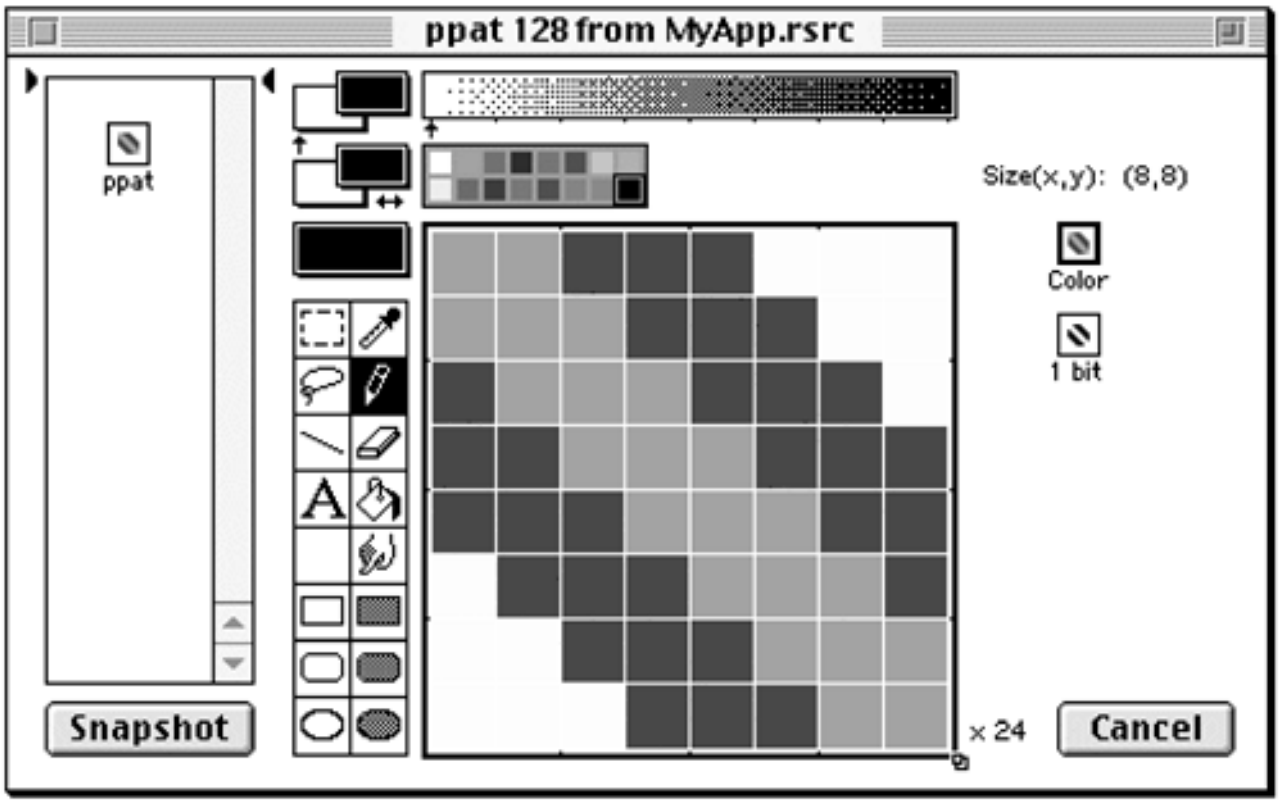
Function	*Description*
`GetPixBounds`	Get the pixel map's boundary rectangle
`GetPixDepth`	Gets the pixel map's pixel depth

Pixel Patterns

The graphics port object stores handles to pixel patterns, structures of type `PixPat`.

Pixel patterns, which define a repeating design, can use colors at any pixel depth, and can be of any width and height that is a power of 2. You can create your own pixel patterns in your program code, but it is usually more convenient to store them in resources of type `'ppat'`. Fig 6 shows an 8-by-8 pixel `'ppat'` resource being created using Resorcerer.

Figure 6. CREATING A 'ppat' RESOURCE USING RESORCERER

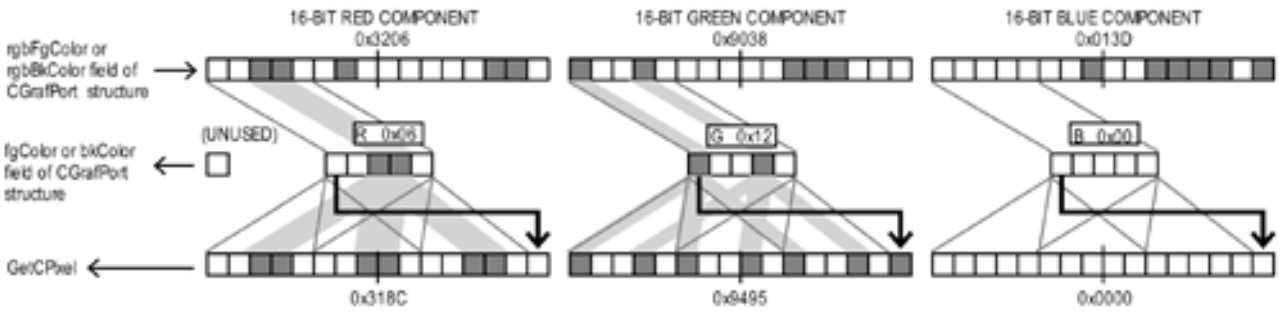


Bit Patterns

Bit patterns date from the era of the black-and-white Macintosh, but may be stored in a color graphics port object. (`PixPat` structures can contain bit patterns as well as pixel patterns.) Bit patterns are defined in data structures of type `Pattern`, a 64-pixel image of a repeating design organized as an 8-by-8 pixel square.

Five bit patterns are predefined as QuickDraw global variables. The five predefined patterns are available not only through the QuickDraw globals but also as system resources. Fig 7 shows images drawn using some of the 38 available system-supplied bit patterns.

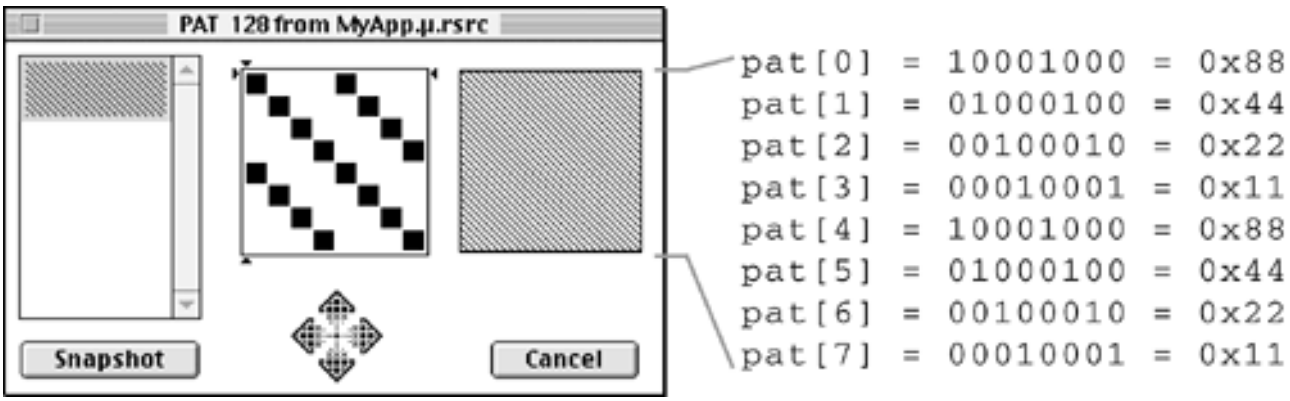
Figure 7. TRANSLATING RGBColor STRUCTURE TO A 16 BIT PIXEL VALUE, AND FROM A 16-BIT PIXEL VALUE TO AN RGBColor STRUCTURE, ON A DIRECT DEVICE



You can create your own bit patterns in your program code, but it is usually more convenient to store them in resources of type `'PAT '` or `'PAT#'`. Fig 8 shows a `'PAT '` resource being created using Resorcerer, together with the contents of the `pat` field of the structure of type `Pattern` that is

created when the resource is loaded.

Figure 8. CREATING A 'PAT' RESOURCE USING RESORCERER



Creating Graphics Ports

Your application creates a (color) graphics port using either the `GetNewCWindow`, `NewCWindow`, or `NewGWorld` function. These functions automatically call `CreateNewPort`, which opens the port.

Translation of RGB Colors to Pixel Values

As previously stated, the graphics port object contains a pointer to the beginning of the onscreen pixel image. When your application specifies an RGB color for a pixel in the pixel image, QuickDraw translates that color into a value appropriate for display on the user's screen. QuickDraw stores this value in the pixel. The **pixel value** is a number used by system software and a graphics device to represent a color. The translation from the color you specify in an `RGBColor` structure to a pixel value is performed at the time you draw the color. The process differs for direct and indexed devices as follows:

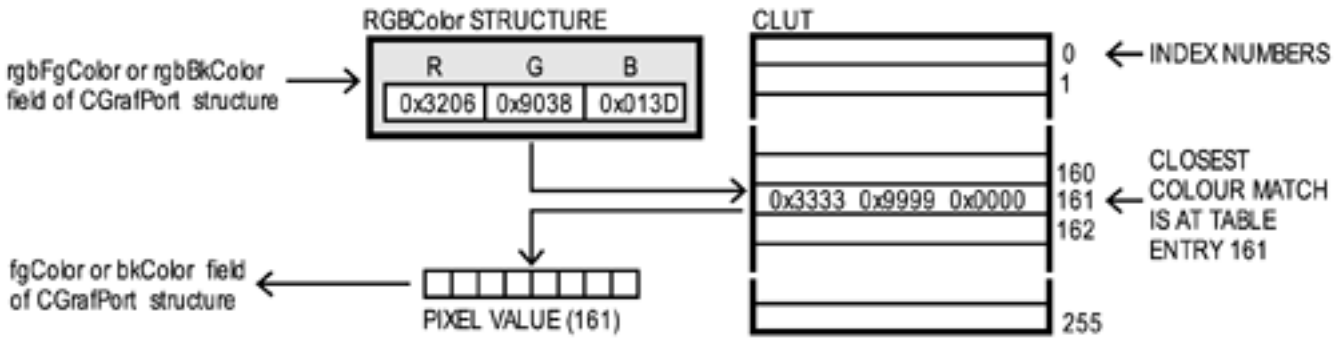
- When drawing on indexed devices, QuickDraw calls the Color Manager to supply the index to the color that most closely matches the requested color in the current device's CLUT. This index becomes the pixel value for that color.
- When drawing on direct devices, QuickDraw truncates the least significant bits from the red, green and blue fields of the `RGBColor` structure. The result becomes the pixel value that QuickDraw sends to the graphics device.

Your application never needs to handle pixel values. However, to clarify the relationship between `RGBColor` structures and the pixels that are actually displayed, the following presents some examples of the derivation of pixel values from `RGBColor` structures.

Derivation of Pixel Values on Indexed Devices

Fig 9 shows the translation of an `RGBColor` structure to an 8-bit pixel value on an indexed device.

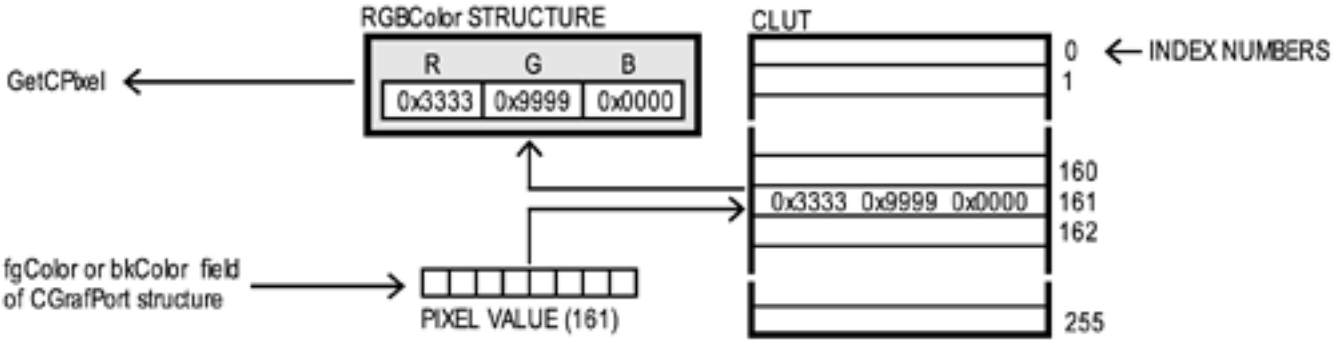
Figure 9. TRANSLATING AN RGBColor STRUCTURE TO AN 8-BIT PIXEL VALUE ON AN INDEXED DEVICE



An application might call `GetCPixel` to determine the color of a pixel set by the pixel value in Fig 9. As shown in Fig 10, the Color Manager uses the pixel value (an index number) to find the `RGBColor` structure stored in the CLUT for that pixel's color. This is the color returned by

`GetCPixel`. As shown in Fig 10, this is not necessarily the exact color first specified.

Figure 10. TRANSLATING 8-BIT PIXEL VALUE ON AN IDEXED DEVICE TO AN RGBColor STRUCTURE



Derivation of Pixel Values on Direct Devices

Fig 11 shows how QuickDraw converts an `RBGColor` structure into a 16-bit pixel value on a direct device. The most significant 5 bits of each field of the `RGBColor` structure are stored in the lower 15 bits of the pixel value. The high bit is unused. Fig 11 also shows how QuickDraw expands a 16-bit pixel value to a 48-bit `RGBColor` structure. Each 5-bit component, and the most significant bit, are inserted into each 16-bit field of the `RGBColor` structure. Note the difference between the result and the original 48-bit value.

Figure 11. TRANSLATING RGBColor STRUCTURE TO A 16 BIT PIXEL VALUE, AND FROM A 16-BIT PIXEL VALUE TO AN RGBColor STRUCTURE, ON A DIRECT DEVICE

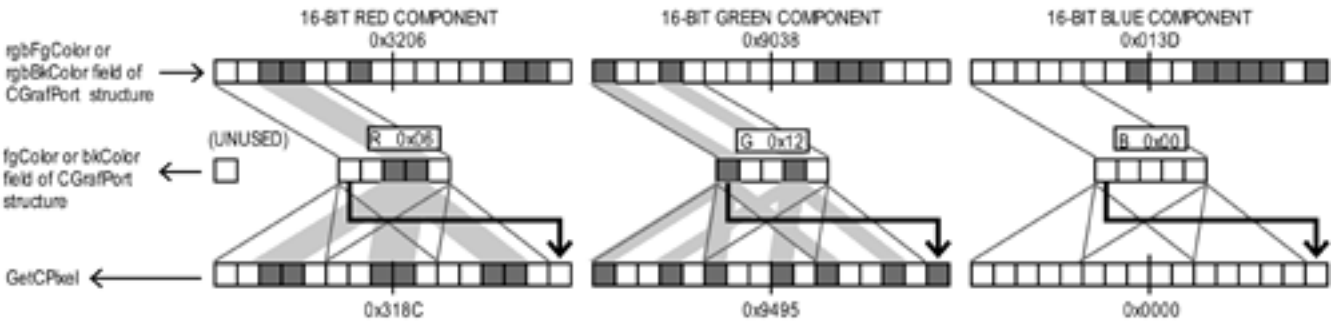
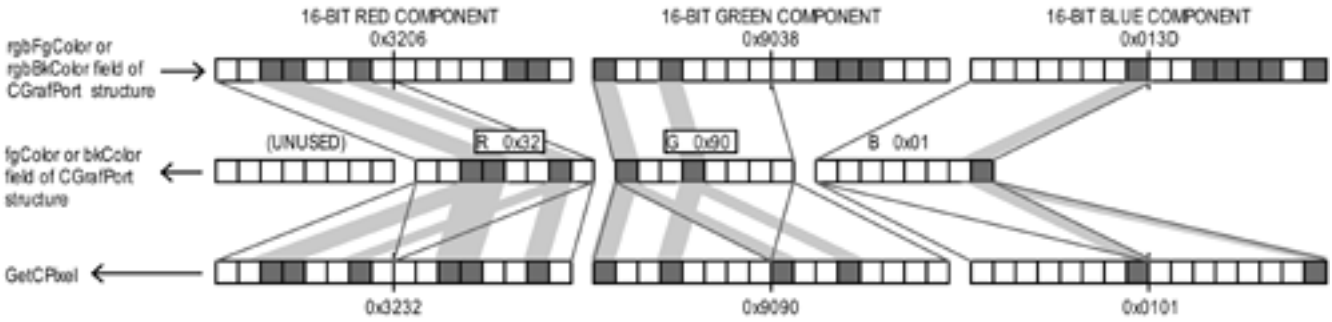


Fig 12 shows how QuickDraw converts an `RBGColor` structure into a 32-bit pixel value on a direct device. The most significant 8 bits of each 16-bit field of the `RGBColor` structure are stored in the lower 3 bytes of the pixel value. 8 bits in the high byte of the pixel value are unused. Fig 12 also shows how QuickDraw expands a 32-bit pixel value to an `RBGColor` structure. Each of the 8-bit components is doubled. Note the difference between the result and the original 48-bit value.

Figure 12. TRANSLATING RGBColor STRUCTURE TO A 32 BIT PIXEL VALUE,

AND FROM A 32-BIT PIXEL VALUE TO AN RGBColor STRUCTURE, ON A DIRECT DEVICE



Colors on Grayscale Screens

When QuickDraw displays a color on a grayscale screen, it computes the luminance, or intensity of light, of the desired color and uses that value to determine the appropriate gray value to draw.

A grayscale device can be a graphics device that the user sets to grayscale. For such a graphics device, Color QuickDraw places an evenly spaced set of grays in the graphics device's CLUT.

Graphics Devices and GDevice Structures

As previously stated, QuickDraw provides a device-independent interface. Your application can draw images in the graphics port for a window and QuickDraw automatically manages the path to the screen-even if the user is using multiple screens. QuickDraw communicates with a video device, such as a plug-in video card or a built-in video interface, by automatically creating and managing a structure of type `GDevice`.

Types of Graphics Device

A **graphics device** is anything QuickDraw can draw into. There are three types of graphics device:

- **Video devices,** which control screens.
- **Offscreen graphics worlds.** (See Chapter 13.)
- **Printing graphics ports.** (See Chapter 15.)

In the case of a video device for an offscreen graphics world, QuickDraw automatically creates, and stores state information in, a `GDevice` structure.

GDevice Structure

QuickDraw creates and initializes a `GDevice` structure for each video device found during startup. QuickDraw also automatically creates a `GDevice` structure when you call `NewGWorld` to create an offscreen graphics world.

A list called a **device list** links together all existing `GDevice` structures. The **current device,** which is sometimes called the **active device,** is that device in the device list into which drawing is currently taking place.

Your application generally never needs to create `GDevice` structures; however, in may need to examine `GDevice` structures to determine the capabilities of the user's screens. The `GDevice` structure is as follows:

```
struct GDevice
{
  short        gdRefNum;      // Reference Number of Driver.
  short        gdID;          // Client ID for search procedures.
  short        gdType;        // Type of device (indexed or direct).
  ITabHandle   gdITable;      // Handle to inverse lookup table for Color Manager.
  short        gdResPref;     // Preferred resolution.
  SProcHndl    gdSearchProc;  // Handle to list of search functions.
  CProcHndl    gdCompProc;    // Handle to list of complement functions.
  short        gdFlags;       // Graphics device flags.
  PixMapHandle gdPMap;        // Handle to pixel map for displayed image.
  long         gdRefCon;      // Reference value.
  Handle       gdNextGD;      // Handle to next GDevice structure.
  Rect         gdRect;        // Device's global boundaries.
  long         gdMode;        // Device's current mode.
  short        gdCCBytes;     // Width of expanded cursor data.
  short        gdCCDepth;     // Depth of expanded cursor data.
  Handle       gdCCXData;     // Handle to cursor's expanded data.
  Handle       gdCCXMask;     // Handle to cursor's expanded mask.
  long         gdReserved;    // (Reserved. Must be 0.)
} ;
typedef struct GDevice GDevice;
```

```
typedef GDevice *GDPtr, **GDHandle;
```

Main Field Descriptions

`gdType` The type of graphics device. The flag bits of this field are as follows:

Constant	*Bit*	*Meaning If Set*
`clutType`	`0`	CLUT device
`fixedType`	`1`	Fixed CLUT device
`directType`	`2`	Direct device

`gdITable` Points to an **inverse table.** This is a special Color Manager data structure that allows index numbers in a CLUT to be found very quickly.

`gdFlags` Device attributes (that is, whether the device is a screen, whether it is the main screen, whether it is set to black-and-white or color, whether it is the active device, and so on.). The main flag bits in this field are as follows:

Constant	*Bit*	*Meaning If Set*
`gdDevType`	`0`	Device is a color device. (If not set, device is a black-and-white device.)
`mainScreen`	`11`	Device is the main screen.
`screenDevice`	`13`	Device is a screen device.
`screenActive`	`15`	Device is current device.

`gdPMap` A handle to the pixel map (`PixMap`) structure.

`gdNextGD` A handle to the next device in the device list. Contains 0 if this is the last graphics device in the device list.

`gdRect` The boundary rectangle of this graphics device. The upper-left corner of the boundary rectangle for the main screen is set to (`0,0`) and all other graphics devices are relative to this.

Setting a Device's Pixel Depth

The `gdPMap` field of the `GDevice` structure contains a handle to a `PixMap` structure which, in turn, contains the `PixelSize` field to which is assigned the pixel depth of the device.

The user can change the pixel depth of video devices. Accordingly, although your application may have a preferred pixel depth, it should be flexible enough to accommodate other pixel depths.

Your application can change the pixel depth using `SetDepth`. However, before calling this function, you should call the `HasDepth` function to confirm that the hardware can support the desired pixel depth. Generally speaking, you should not change pixel depth without first seeking the consent of the user via an alert or dialog.

Other Graphics Managers

In addition to the QuickDraw functions, several other collections of system software functions are available to assist you in drawing images.

Palette Manager

Your application can use the Palette Manager to provide more sophisticated color support on indexed graphics devices. The Palette Manager allows your application to specify sets of colors that it needs on a window-by-window basis.

Color Picker Utilities

To solicit color choices from users, your application can use the Color Picker Utilities. The Color Picker Utilities also provide functions that allow your application to convert between colors specified in `RGBColor` structures and colors specified for other color models, such as the CMYK (cyan, magenta, yellow, black) model used for many color printers (see Chapter 25).

Coping with Multiple Monitors

Aspects of coping with a multiple monitors environment are addressed in Chapter 25.

Relevant QuickDraw Constants, Data Types, and Functions

Constants

Flag Bits of gdType Field of GDevice Structure

```
clutType     = 0
fixedType    = 1
directType   = 2
```

Flag Bits of gdFlags Field of GDevice Structure

```
gdDevType    = 0
burstDevice  = 7
ext32Device  = 8
ramInit      = 10
mainScreen   = 11
allInit      = 12
screenDevice = 13
noDriver     = 14
screenActive = 15
```

Pixel Type

```
RGBDirect     = 16  16 and 32 bits-per-pixel pixelType value.
```

Data Types

```
typedef struct OpaqueGrafPtr* GrafPtr;
typedef GrafPtr CGrafPtr;
```

Pixel Map

```
struct PixMap
{
  Ptr          baseAddr;     // Pointer to image data.
  short        rowBytes;     // Flags, and bytes in a row.
  Rect         bounds;       // Boundary rectangle.
  short        pmVersion;    // Pixel Map version number.
  short        packType;     // Packing format.
  long         packSize;     // Size of data in packed state.
  Fixed        hRes;         // Horizontal resolution in dots per inch.
  Fixed        vRes;         // Vertical resolution in dots per inch.
  short        pixelType;    // Format of pixel image.
  short        pixelSize;    // Physical bits per pixel.
  short        cmpCount;     // Number of components in each pixel.
  short        cmpSize;      // Number of bits in each component.
  long         planeBytes;   // Offset to next plane.
  CTabHandle pmTable;        // Handle to a color table for this image.
  long  pmReserved;          // (Reserved.)
} ;
typedef struct PixMap PixMap,*PixMapPtr,**PixMapHandle;
```

BitMap

```
struct BitMap
{
  Ptr          baseAddr;      // Pointer to bit image.
  short        rowBytes;      // Row width.
  Rect         bounds;        // Boundary rectangle.
};
typedef struct BitMap BitMap;
typedef BitMap *BitMapPtr, **BitMapHandle;
```

Pixel Pattern

```
struct PixPat
{
  short        patType;       // Type of pattern.
  PixMapHandle patMap;        // The pattern's pixel map.
  Handle       patData;       // Pixel map's data.
  Handle       patXData;      // Expanded Pattern data (internal use).
  short        patXValid;     // Flags whether expanded Pattern valid.
  Handle       patXMap;       // Handle to expanded Pattern data (reserved).
  Pattern      pat1Data;      // Bit map's data.
};
typedef struct PixPat PixPat;
typedef PixPat *PixPatPtr;
typedef PixPatPtr *PixPatHandle;
```

Pattern

```
struct Pattern
{
  UInt8        pat[8];
};
typedef struct Pattern Pattern;
typedef Pattern *PatPtr;
typedef PatPtr *PatHandle;
```

GDevice

```
struct GDevice
{
  short        gdRefNum;      // Reference Number of Driver.
  short        gdID;          // Client ID for search procedures.
  short        gdType;        // Type of device (indexed or direct).
  ITabHandle   gdITable;      // Handle to inverse lookup table for Color Manager.
  short        gdResPref;     // Preferred resolution.
  SProcHndl    gdSearchProc;  // Handle to list of search functions.
  CProcHndl    gdCompProc;    // Handle to list of complement functions.
  short        gdFlags;       // Graphics device flags.
  PixMapHandle gdPMap;        // Handle to pixel map for displayed image.
  long         gdRefCon;      // Reference value.
  Handle       gdNextGD;      // Handle to next GDevice structure.
  Rect         gdRect;        // Device's global boundaries.
  long         gdMode;        // Device's current mode.
  short        gdCCBytes;     // Width of expanded cursor data.
  short        gdCCDepth;     // Depth of expanded cursor data.
  Handle       gdCCXData;     // Handle to cursor's expanded data.
```

```
  Handle       gdCCXMask;     // Handle to cursor's expanded mask.
  long         gdReserved;    // (Reserved. Must be 0.)
};
typedef struct GDevice GDevice;
typedef GDevice *GDPtr, **GDHandle;
```

Functions

Opening and Closing Graphics Ports

```
CgrafPtr      CreateNewPort(void);
Void          DisposePort(CGrafPtr port);
```

Saving and Restoring Graphics Ports

```
void          GetPort(GrafPtr *port);
void          SetPort(GrafPtr port);
void          SetPortDialogPort(DialogPtr dialog);
void          SetPortWindowPort(WindowRef window);
```

Getting a Pointer to the Owning Window

```
WindowRef     GetWindowFromPort(CGrafPtr port);
```

Graphics Port Accessors

```
PixMapHandle  GetPortPixMap(CGrafPtr port);
Rect          GetPortBounds(CGrafPtr port,Rect *rect);
void          SetPortBounds(CGrafPtr port,const Rect *rect);
RgnHandle     GetPortVisibleRegion(CGrafPtr port,RgnHandle visRgn);
void          SetPortVisibleRegion(CGrafPtr port,RgnHandle visRgn);
RgnHandle     GetPortClipRegion(CGrafPtr port,RgnHandle clipRgn);
void          SetPortClipRegion(CGrafPtr port,RgnHandle clipRgn);
void          SetClip(RgnHandle rgn);
RGBColor      GetPortForeColor(CGrafPtr port,RGBColor *foreColor);
void          RGBForeColor(const RGBColor *color);
RGBColor      GetPortBackColor(CGrafPtr port,RGBColor *backColor);
void          RGBBackColor(const RGBColor *color);
PixPatHandle  GetPortBackPixPat(CGrafPtr port,PixPatHandle backPattern);
void          SetPortBackPixPat(CGrafPtr port,PixPatHandle backPattern);
void          BackPat(const Pattern *pat);
PixPatHandle  GetPortPenPixPat(CGrafPtr port,PixPatHandle penPattern);
void          SetPortPenPixPat(CGrafPtr port,PixPatHandle penPattern);
void          PenPat(const Pattern *pat);
PixPatHandle  GetPortFillPixPat(CGrafPtr port,PixPatHandle fillPattern);
Point         GetPortPenLocation(CGrafPtr port,Point *penLocation);
void          MoveTo(short h,short v);
Point         GetPortPenSize(CGrafPtr port,Point *penSize);
void          SetPortPenSize(CGrafPtr port,Point penSize);
void          PenSize(short width,short height);
SInt32        GetPortPenMode(CGrafPtr port);
void          SetPortPenMode(CGrafPtr port,SInt32 penMode);
void          PenMode(short mode);
short         GetPortTextFont(CGrafPtr port);
void          TextFont(short font);
void          HidePen(void);
void          ShowPen(void);
```

```
short         GetPortPenVisibility(CGrafPtr port);
short         GetPortTextSize(CGrafPtr port);
void          TextSize(short size);
Style         GetPortTextFace(CGrafPtr port);
void          TextFace(StyleParameter face);
short         GetPortTextMode(CGrafPtr port);
void          TextMode(short mode)
RGBColor      GetPortHiliteColor(CGrafPtr port,RGBColor *hiliteColor);
```

Creating, Setting, Disposing of, and Accessing Pixel Maps

```
PixMapHandle  NewPixMap(void);
void          CopyPixMap(PixMapHandle srcPM,PixMapHandle dstPM);
void          SetPortPix(PixMapHandle pm);
void          DisposePixMap(PixMapHandle pm);
Rect          GetPixBounds(PixMapHandle pixMap,Rect *bounds);
short         GetPixDepth(PixMapHandle pixMap);
```

Creating, Setting and Disposing of Graphics Device Structures

```
GDHandle      NewGDevice(short refNum,long mode);
void          InitGDevice(short qdRefNum,long mode,GDHandle gdh);
void          SetDeviceAttribute(GDHandle gdh,short attribute,Boolean value);
void          SetGDevice(GDHandle gd);
void          DisposeGDevice(GDHandle gdh);
```

Getting the Available Graphics Devices

```
GDHandle      GetGDevice(void);
GDHandle      GetMainDevice(void);
GDHandle      GetNextDevice(GDHandle curDevice);
GDHandle      GetDeviceList(void);
```

Determining the Characteristics of a Video Device

```
Boolean       TestDeviceAttribute(GDHandle gdh,short attribute);
void          ScreenRes(short *scrnHRes,short *scrnVRes);
```

Changing the Pixel Depth of a Video Device

```
OSErr         SetDepth(GDHandle gd,short depth,short whichFlags,short flags);
short         HasDepth(GDHandle gd,short depth,short whichFlags,short flags);
```

Demonstration Program PreQuickDraw Listing

```
//
****************************************************************************************
// PreQuickDraw.c                                                        CLASSIC EVENT
MODEL
//
****************************************************************************************
//
// This program opens a window in which is displayed some information retrieved from the
// GDevice structure for the main video device, from the graphics port's pixel map, and from
// the graphics port object using QuickDraw functions.
//
// A Demonstration menu allows the user to set the monitor to various pixel depths and to
// restore the original pixel depth. Setting the monitor to a pixel depth of 8 (256 colors)
// or less causes the colors in the color table to be displayed.
//
// The program utilises 'plst', 'MBAR', 'MENU', 'WIND', and 'STR#' resources, and a 'SIZE'
// resource with the acceptSuspendResumeEvents, canBackground, doesActivateOnFGSwitch, and
// isHighLevelEventAware flags set.
//
//
****************************************************************************************

// ..........................................................................
includes

#include <Carbon.h>

// ..........................................................................
defines

#define rMenubar              128
#define rWindow               128
#define mAppleApplication     128
#define  iAbout               1
#define mFile                 129
#define  iQuit                12
#define mDemonstration        131
#define  iSetDepth8           1
#define  iSetDepth16          2
#define  iSetDepth32          3
#define  iRestoreStartDepth   5
#define rIndexedStrings       128
#define  sMonitorInadequate   1
#define  sMonitorAtThatDepth  2
#define  sMonitorAtStartDepth 3
#define  sRestoringMonitor    4
#define MAX_UINT32            0xFFFFFFFF

// ..................................................................... global
variables

Boolean gDone;
SInt16  gStartupPixelDepth;

// ..................................................................... function
prototypes

void    main                    (void);
void    doPreliminaries         (void);
OSErr   quitAppEventHandler     (AppleEvent *,AppleEvent *,SInt32);
```

```
void    doEvents                  (EventRecord *);
void    doDisplayInformation      (WindowRef);
Boolean doCheckMonitor            (void);
void    doSetMonitorPixelDepth    (SInt16);
void    doRestoreMonitorPixelDepth (void);
void    doMonitorAlert            (Str255);

// ****************************************************************************************
main

void  main(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  WindowRef     windowRef;
  SInt16        entries = 0;
  Str255        theString;
  EventRecord   EventStructure;

  // ..................................................................... do
preliminaries

  doPreliminaries();

  // ............................................................. set up menu bar and
menus

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    ExitToShell();
  SetMenuBar(menubarHdl);
  DrawMenuBar();

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
      DisableMenuItem(menuRef,0);
    }
  }

  // ..................................... check if monitor can display at least 16-bit
color

  if(!doCheckMonitor())
  {
    GetIndString(theString,rIndexedStrings,sMonitorInadequate);
    doMonitorAlert(theString);
  }
  // ..................................... open windows, set font, show windows, move
windows

  if(!(windowRef = GetNewCWindow(rWindow,NULL,(WindowRef)-1)))
    ExitToShell();

  SetPortWindowPort(windowRef);
  TextSize(10);

  // ..................................................................... enter
eventLoop
```

```
  gDone = false;

  while(!gDone)
  {
    if(WaitNextEvent(everyEvent,&EventStructure,MAX_UINT32,NULL))
      doEvents(&EventStructure);
  }
}

// ************************************************************************
doPreliminaries

void  doPreliminaries(void)
{
  OSErr osError;

  MoreMasterPointers(32);
  InitCursor();
  FlushEvents(everyEvent,0);

  osError = AEInstallEventHandler(kCoreEventClass,kAEQuitApplication,
                            NewAEEventHandlerUPP((AEEventHandlerProcPtr)
quitAppEventHandler),
                            0L,false);

  if(osError != noErr)
    ExitToShell();
}

// ************************************************************************
doQuitAppEvent

OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  OSErr    osError;
  DescType returnedType;
  Size     actualSize;

  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,
NULL,0,
                              &actualSize);

  if(osError == errAEDescNotFound)
  {
    gDone = true;
    osError = noErr;
  }
  else if(osError == noErr)
    osError = errAEParamMissed;
  return osError;
}

// ************************************************************************
doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
  SInt32         menuChoice;
  MenuID         menuID;
  MenuItemIndex  menuItem;
  WindowPartCode partCode;
  WindowRef      windowRef;
  Rect           portRect;

  switch(eventStrucPtr->what)
  {
```

```
case kHighLevelEvent:
  AEProcessAppleEvent(eventStrucPtr);
  break;

case keyDown:
  if((eventStrucPtr->modifiers & cmdKey) != 0)
  {
    menuChoice = MenuEvent(eventStrucPtr);
    menuID = HiWord(menuChoice);
    menuItem = LoWord(menuChoice);
    if(menuID == mFile && menuItem  == iQuit)
      gDone = true;
  }
  break;

case mouseDown:
  if(partCode = FindWindow(eventStrucPtr->where,&windowRef))
  {
    switch(partCode)
    {
      case inMenuBar:
        menuChoice = MenuSelect(eventStrucPtr->where);
        menuID = HiWord(menuChoice);
        menuItem = LoWord(menuChoice);

        if(menuID == 0)
          return;

        switch(menuID)
        {
          case mAppleApplication:
            if(menuItem == iAbout)
              SysBeep(10);
            break;

          case mFile:
            if(menuItem == iQuit)
              gDone = true;
            break;
          case mDemonstration:
            if(menuItem == iSetDepth8)
              doSetMonitorPixelDepth(8);
            else if(menuItem == iSetDepth16)
              doSetMonitorPixelDepth(16);
            else if(menuItem == iSetDepth32)
              doSetMonitorPixelDepth(32);
            else if(menuItem == iRestoreStartDepth)
              doRestoreMonitorPixelDepth();
            break;
        }
        HiliteMenu(0);
        break;

      case inDrag:
        DragWindow(windowRef,eventStrucPtr->where,NULL);
        GetWindowPortBounds(windowRef,&portRect);
        InvalWindowRect(windowRef,&portRect);
        break;
    }
  }
  break;

case updateEvt:
  windowRef = (WindowRef) eventStrucPtr->message;
  BeginUpdate(windowRef);
  SetPortWindowPort(windowRef);
```

```
      doDisplayInformation(windowRef);
      EndUpdate(windowRef);
      break;
  }
}

// ************************************************************************
doDisplayInformation

void  doDisplayInformation(WindowRef windowRef)
{
  RGBColor     whiteColor = {  0xFFFF, 0xFFFF, 0xFFFF } ;
  RGBColor     blueColor  = {  0x3333, 0x3333, 0x9999 } ;
  Rect         portRect;
  GDHandle     deviceHdl;
  SInt16       videoDeviceCount = 0;
  Str255       theString;
  SInt16       deviceType, pixelDepth, bytesPerRow;
  Rect         theRect;
  GrafPtr      grafPort;
  PixMapHandle pixMapHdl;
  CTabHandle   colorTableHdl;
  SInt16       entries = 0, vert = 28, horiz = 250, index = 0;
  RGBColor     getPixelColor,colorTableColor;

  RGBForeColor(&whiteColor);
  RGBBackColor(&blueColor);
  GetWindowPortBounds(windowRef,&portRect);
  EraseRect(&portRect);
  QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);

  // ..................................................................... Get Device
List

  deviceHdl = GetDeviceList();

  // ..................................................... count video devices in device
list

  while(deviceHdl != NULL)
  {
    if(TestDeviceAttribute(deviceHdl,screenDevice))
      videoDeviceCount ++;

    deviceHdl = GetNextDevice(deviceHdl);
  }

  NumToString(videoDeviceCount,theString);
  MoveTo(10,20);
  DrawString(theString);
  if(videoDeviceCount < 2)
    DrawString("\p video device in the device list.");
  else
    DrawString("\p video devices in the device list.");

  // ..................................................................... Get Main
Device

  deviceHdl = GetMainDevice();

  // ..................................................................... determine device
type

  MoveTo(10,35);

  if(((1 << gdDevType) & (*deviceHdl)->gdFlags) != 0)
```

```
    DrawString("\pThe main video device is a color device.");
  else
    DrawString("\pThe main video device is a monochrome device.");

  MoveTo(10,50);
  deviceType = (*deviceHdl)->gdType;
  switch(deviceType)
  {
    case clutType:
      DrawString("\pIt is an indexed device with variable CLUT.");
      break;

    case fixedType:
      DrawString("\pIt is is an indexed device with fixed CLUT.");
      break;

    case directType:
      DrawString("\pIt is a direct device.");
      break;
  }
  // ..................................................................... Get Handle to Pixel
Map

  grafPort = GetWindowPort(windowRef);
  pixMapHdl = GetPortPixMap(grafPort);
  // pixMapHdl = (*deviceHdl)->gdPMap; // alternative method

  // ..................................................................... get and display pixel
depth

  MoveTo(10,70);
  DrawString("\pPixel depth = ");

  pixelDepth = GetPixDepth(pixMapHdl);
  // pixelDepth = (*(*deviceHdl)->gdPMap)->pixelSize;  // alternative method

  NumToString(pixelDepth,theString);
  DrawString(theString);

  // ..................................................................... get and display bytes per
row

  MoveTo(10,90);
  bytesPerRow = (*pixMapHdl)->rowBytes & 0x7FFF;
  DrawString("\pBytes per row = ");
  NumToString(bytesPerRow,theString);
  DrawString(theString);

  // ..................................................... Get Device's Global Boundary
Rectangle

  theRect = (*deviceHdl)->gdRect;

  // ..................................... calculate and display total pixel image
bytes

  MoveTo(10,105);
  DrawString("\pTotal pixel image bytes = ");
  NumToString(bytesPerRow * theRect.bottom,theString);
  DrawString(theString);

  // ..................................................... display device's boundary
rectangle

  MoveTo(10,130);
  TextFace(bold);
```

```
  DrawString("\pGraphics Device's Boundary Rectangle");
  TextFace(normal);
  MoveTo(10,145);
  DrawString("\p(gdRect field of GDevice structure)");

  MoveTo(10,160);
  DrawString("\pBoundary rectangle top = ");
  NumToString(theRect.top,theString);
  DrawString(theString);

  MoveTo(10,175);
  DrawString("\pBoundary rectangle left = ");
  NumToString(theRect.left,theString);
  DrawString(theString);
  MoveTo(10,190);
  DrawString("\pBoundary rectangle bottom = ");
  NumToString(theRect.bottom,theString);
  DrawString(theString);

  MoveTo(10,205);
  DrawString("\pBoundary rectangle right = ");
  NumToString(theRect.right,theString);
  DrawString(theString);

  // ..................................... Get and Display Pixel Map's Boundary
Rectangle

  GetPixBounds(pixMapHdl,&theRect);

  MoveTo(10,225);
  TextFace(bold);
  DrawString("\pPixel Map's Boundary Rectangle");
  TextFace(normal);
  MoveTo(10,240);
  DrawString("\p(bounds field of PixMap structure)");

  MoveTo(10,255);
  DrawString("\pBoundary rectangle top = ");
  NumToString(theRect.top,theString);
  DrawString(theString);

  MoveTo(10,270);
  DrawString("\pBoundary rectangle left = ");
  NumToString(theRect.left,theString);
  DrawString(theString);

  MoveTo(10,285);
  DrawString("\pBoundary rectangle bottom = ");
  NumToString(theRect.bottom,theString);
  DrawString(theString);

  MoveTo(10,300);
  DrawString("\pBoundary rectangle right = ");
  NumToString(theRect.right,theString);
  DrawString(theString);

  MoveTo(10,320);
  DrawString("\pOn Mac OS X, drag window after pixel depth and screen resolution changes
to");
  DrawString("\p ensure that");
  MoveTo(10,333);
  DrawString("\pbytes per row, pixel image bytes, and color values are updated.");

  // ......................... Get and Display RGB Components of Requested Background
Color
```

```
  MoveTo(250,255);
  GetBackColor(&blueColor);
  DrawString("\pRequested background color (rgb) = ");
  MoveTo(250,270);
  NumToString(blueColor.red,theString);
  DrawString(theString);
  DrawString("\p  ");
  NumToString(blueColor.green,theString);
  DrawString(theString);
  DrawString("\p  ");
  NumToString(blueColor.blue,theString);
  DrawString(theString);

  // ........ If Direct Device, Get and Display RGB Components of Color Returned by
GetCPixel

  if(deviceType == directType)
  {
    MoveTo(250,285);
    GetCPixel(10,10,&getPixelColor);
    DrawString("\pColor returned by CetCPixel (rgb) = ");
    MoveTo(250,300);
    NumToString(getPixelColor.red,theString);
    DrawString(theString);
    DrawString("\p  ");
    NumToString(getPixelColor.green,theString);
    DrawString(theString);
    DrawString("\p  ");
    NumToString(getPixelColor.blue,theString);
    DrawString(theString);
  }

  // ............................................ else prepare to display color table
index

  else
  {
    MoveTo(250,285);
    DrawString("\pBackground color (color table index):");
  }

  // ............................................................ Get Handle to Color
Table

  colorTableHdl = (*pixMapHdl)->pmTable;

  // ..................................... if any entries in color table, draw the colors

  MoveTo(250,20);
  DrawString("\pColor table:");

  entries = (*colorTableHdl)->ctSize;

  if(entries < 2)
  {
    MoveTo(260,100);
    DrawString("\pOnly one (dummy) entry in the color");
    MoveTo(260,115);
    DrawString("\ptable. To cause the color table to be");
    MoveTo(260,130);
    DrawString("\pbuilt, set the monitor to bit depth 8");
    MoveTo(260,145);
    DrawString("\p(256 colors), causing it to act like ");
    MoveTo(260,160);
    DrawString("\pan indexed device.");
    SetRect(&theRect,250,28,458,236);
```

```
    FrameRect(&theRect);
  }

  for(index = 0;index <= entries;index++)
  {
    SetRect(&theRect,horiz,vert,horiz+12,vert+12);
    colorTableColor = (*colorTableHdl)->ctTable[index].rgb;
    RGBForeColor(&colorTableColor);
    PaintRect(&theRect);

    // .... also, if device is not a  direct device, and current color matches
background ...

    if(deviceType == clutType || deviceType == fixedType)
    {
      if(colorTableColor.red == blueColor.red &&
         colorTableColor.green == blueColor.green &&
         colorTableColor.blue == blueColor.blue)
      {

        // ..................... outline the drawn color and display the color table index

        RGBForeColor(&whiteColor);
        InsetRect(&theRect,-1,-1);
        FrameRect(&theRect);
        MoveTo(250,300);
        NumToString(index,theString);
        DrawString(theString);
      }
    }

    horiz += 13;
    if(horiz > 445)
    {
      horiz = 250;
      vert += 13;
    }
  }

  QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
}

// ************************************************************************
doCheckMonitor

Boolean doCheckMonitor(void)
{
  GDHandle  mainDeviceHdl;

  mainDeviceHdl = GetMainDevice();
  if(!(HasDepth(mainDeviceHdl,16,gdDevType,1)))
  {
    DisableMenuItem(GetMenuRef(mDemonstration),0);
    return false;
  }
  else
  {
    gStartupPixelDepth = (**((**mainDeviceHdl).gdPMap)).pixelSize;
    return true;
  }
}

// ********************************************************************
doSetMonitorPixelDepth

void  doSetMonitorPixelDepth(SInt16 requiredDepth)
```

```
{
  GDHandle mainDeviceHdl;
  Str255   alertString;
  SInt16   currentPixelDepth;

  mainDeviceHdl = GetMainDevice();
  currentPixelDepth = (**((**mainDeviceHdl).gdPMap)).pixelSize;

  if(currentPixelDepth != requiredDepth)
  {
    SetDepth(mainDeviceHdl,requiredDepth,gdDevType,1);
  }
  else
  {
    GetIndString(alertString,rIndexedStrings,sMonitorAtThatDepth);
    doMonitorAlert(alertString);
  }
}

// ************************************************************
doRestoreMonitorPixelDepth

void  doRestoreMonitorPixelDepth(void)
{
  GDHandle mainDeviceHdl;
  Str255   alertString;
  SInt16   pixelDepth;

  mainDeviceHdl = GetMainDevice();
  pixelDepth = (**((**mainDeviceHdl).gdPMap)).pixelSize;

  if(pixelDepth != gStartupPixelDepth)
  {
    GetIndString(alertString,rIndexedStrings,sRestoringMonitor);
    doMonitorAlert(alertString);
    SetDepth(mainDeviceHdl,gStartupPixelDepth,gdDevType,1);
  }
  else
  {
    GetIndString(alertString,rIndexedStrings,sMonitorAtStartDepth);
    doMonitorAlert(alertString);
  }
}

// ************************************************************************
doMonitorAlert

void  doMonitorAlert(Str255 labelText)
{
  SInt16 itemHit;
  StandardAlert(kAlertNoteAlert,labelText,NULL,NULL,&itemHit);
}

//
**********************************************************************************************
```

Demonstration Program PreQuickDraw Comments

When this program is run, the user should

- Drag the window to various positions on the main screen, noting, on Mac OS 8/9 only, the changes to the coordinates of the pixel map's boundary rectangle. (On Mac OS X these coordinates represent the bounds of the Core Graphics window that backs the Carbon window, not the screen.)

- Change between the available monitor resolutions, noting the changes in the bytes per row and total pixel image bytes figures displayed in the window.

- Using the Demonstration menu, change between the available pixel depths, noting the changes to the pixel depth and total pixel image bytes figures, and the background color values, displayed in the window.

- Note that, when a pixel depth of 8 is set on a direct device, the device creates a CLUT and operates like a direct device. In this case, the background color value is the color table entry (index), and the relevant color in the color table display is framed in white.

On Mac OS 8/9, if the user's monitor is set to thousands or millions of colors when the program is run for the first time, the color table will not be built. It will be built when the user first sets the pixel depth to 8 (256 colors).

main

The call to `doCheckMonitor` determines whether the monitor can support a pixel depth of at least 16. If it cannot, the Demonstration menu is disabled, false is returned, and an alert is displayed advising the user that the Demonstration menu will be unavailable. If the monitor can support a pixel depth of at least 16, the current pixel depth is assigned to the global variable `gStartupPixelDepth`.

doEvents

In the case of a mouse-down event, in the `inDrag` case, when the user releases the mouse button, the window is invalidated, causing it to be redrawn.

doDisplayInformation

At the first two lines, RGB colors are assigned to the window's graphics port's `rgbFgColor` and U fields. The call to `EraseRect` causes the content region to be filled with the background color.

Get Device List

The call to `GetDeviceList` gets a handle to the first `GDevice` structure in the device list. The device list is then "walked" in the while loop. For every video device found in the list, the variable `videoDeviceCount` is incremented. `GetNextDevice` gets a handle to the next device in the device list.

Get Main Device

`GetMainDevice` gets a handle to the startup device, that is, the device on which the menu bar appears.

Following the call to `MoveTo`, the `gdDevType` bit is tested to determine whether the main (startup) device is a color or black-and-white device.

In the next block, the `gdType` field of the `GDevice` structure is examined to determine whether the device is an indexed device with a variable CLUT, an indexed device with a fixed CLUT, or a direct device (or a direct device set to display 256 colors or less and, as a consequence, acting like an indexed device).

Get Handle to Pixel Map

The call to `GetWindowPort` gets the reference to the window's graphics port required by the call to `GetPortPixMap`. `GetPortPixMap` gets a handle to the pixel map. (The following line shows an alternative method of obtaining a handle to a pixel map, in this case from the `GDevice` structure.)

In the next block, `GetPixDepth` is called to get the pixel depth. (The following line shows an alternative method of obtaining the pixel depth, in this case from the `GDevice` structure.)

At the next block, the number of bytes in each row in the pixel map is determined. (The high bit in the `rowBytes` field of the `PixMap` structure is a flag which indicates whether the data structure is a `PixMap` structure or a `BitMap` structure.)

Get Device's Boundary Rectangle

At the first line of this block, the device's boundary rectangle is extracted from the `GDevice` structure's `gdRect` field.

At the next block, the bytes per row value is multiplied by the height of the boundary rectangle to arrive at the total number of bytes in the pixel image.

The boundary rectangle's top, left, bottom, and right coordinates are then drawn in the window.

Get and Display Pixel Map's Boundary Rectangle

The call to `GetPixBounds` gets the pixel map's bounding rectangle. The rectangle's top, left, bottom, and right coordinates are then drawn in the window.

Get and Display RGB Components of Requested Background Color

The second line of this block calls `GetBackColor` to get the graphics port's background color. The red, green, and blue values are then printed in the window.

If Direct Device, Get and Display RGB Components of Color Returned by GetCPixel

If the device is a direct device, `GetCPixel` is called to get the color of a pixel in the window drawn with the background color. The red, green and blue values are then printed in the window.

If the device is not a direct device, some preparatory text is drawn in the window.

Get Handle To Color Table

The first and fourth lines get a handle to the color table in the `GDevice` structure's pixel map and the number of entries in that table. (Note that the `ctSize` field of the `ColorTable` structure contains the number of table entries minus one.)

On Mac OS 8/9, QuickDraw only calls the Color Manager to build the color table if the device is an indexed device (or a direct device acting as an indexed device). Thus, on Mac OS 8/9, there will only be a dummy entry in the color table unless the monitor is an indexed device or a direct device set to display 256 colors or less.

The final block paints small colored rectangles for each entry in the color table. If the main device is an indexed device (or if it is a direct device set to display 256 colors or less), the color table entry being used as the best match for the requested background color is outlined in white and the index value is drawn.

doCheckMonitor

`doCheckMonitor` is called at program start to determine whether the main device

supports at least 16-bit color and, if it does, to assign the main device's pixel depth at startup to the global variable `gStartupPixelDepth`.

The call to `GetMainDevice` gets a handle to the main device's `GDevice` structure. The function `HasDepth` is used to determine whether the device supports at least 16-bit color. If it does not, the Demonstration menu is disabled and false is returned. If it does, the pixel depth is extracted from the `pixelSize` field of the `PixMap` structure in the `GDevice` structure and assigned to the global variable `gStartupPixelDepth`.

doSetMonitorPixelDepth

`doSetMonitorPixelDepth` is called when one of the the first three items in the Demonstration menu is chosen.

If the current pixel depth determined at the first two lines is not equal to the required new depth, `SetDepth` is called to set the main device's pixel depth to the required depth.

If the current pixel depth is equal to the required pixel depth, an alert is displayed advising the user that the device is currently set to that pixel depth.

doRestoreMonitorPixelDepth

`doRestoreMonitorPixelDepth` is called, when the last item in the Demonstration menu is chosen, to reset the main device's pixel depth to the startup pixel depth.

If the current pixel depth determined at the first two lines is not equal to the startup pixel depth, a string is retrieved from a `'STR#'` resource and passed to the function `doMonitorAlert`, which displays a movable modal alert box advising the user that the monitor's bit depth is about to be changed to the startup pixel depth. When the user dismisses the alert box`, SetDepth` sets the main device's pixel depth to the startup pixel depth.

If the current pixel depth is the startup pixel depth, the last two lines display an alert box advising the user that the device is currently set to that pixel depth.

Chapter 12. Drawing with QuickDraw

IN THIS CHAPTER

- Introduction
- The Coordinate Plane, Points, Rectangles, and Regions
- The Graphics Pen, Foreground and Background Colors, Pixel Patterns and Bit Patterns, and Transfer Modes
- Drawing Lines and Framed Shapes
- Drawing Painted and Filled Shapes
- Erasing and Inverting Shapes
- Drawing Pictures
- Drawing Text
- Copying Pixel Images Between Graphics Ports
- Scrolling Pixels in the Port Rectangle
- Manipulating Rectangles and Regions
- Highlighting
- Drawing Other Graphics Entities
- Saving and Restoring the Graphics Port Drawing State
- Main QuickDraw Constants, Data Types and Functions
- Relevant Appearance Manager Data Types and Functions
- Demonstration Program QuickDraw Listing
- Demonstration Program QuickDraw Comments

Introduction

As stated in Chapter 11, QuickDraw is a collection of system software routines that your application uses to perform imaging operations-that is, the construction and display of graphical information for display on output devices such as screens and printers.

The Coordinate Plane, Points, Rectangles, and Regions

The following mathematical constructs are widely used in QuickDraw's functions and data types:

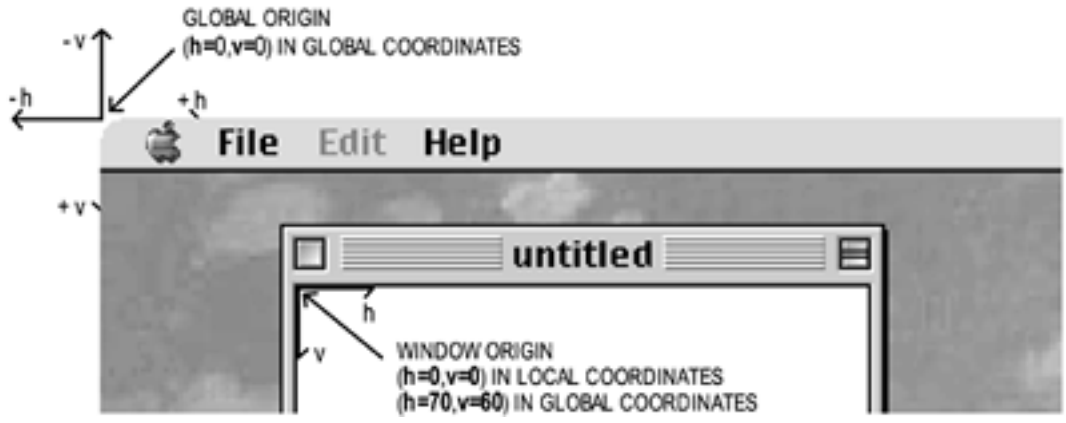
- The coordinate plane
- The point
- The rectangle
- The region

The Coordinate Plane

A Macintosh screen (or screens) represents part of a global coordinate plane bounded by the limits of QuickDraw coordinates (-32768 to 32767). The (0,0) origin point of this global coordinate plane is at the upper-left corner of the main screen. From the upper-left coordinate of the main screen, coordinate values decrease to the left and up and increase to the right and down. Any pixel on the screen can be specified by a vertical coordinate and a horizontal coordinate.

In addition to the global coordinate system, QuickDraw maintains a **local coordinate system** for every window's graphics port. The relationship between local and global coordinates is shown in Fig 1.

Figure 1. LOCAL AND GLOBAL COORDINATE SYSTEMS

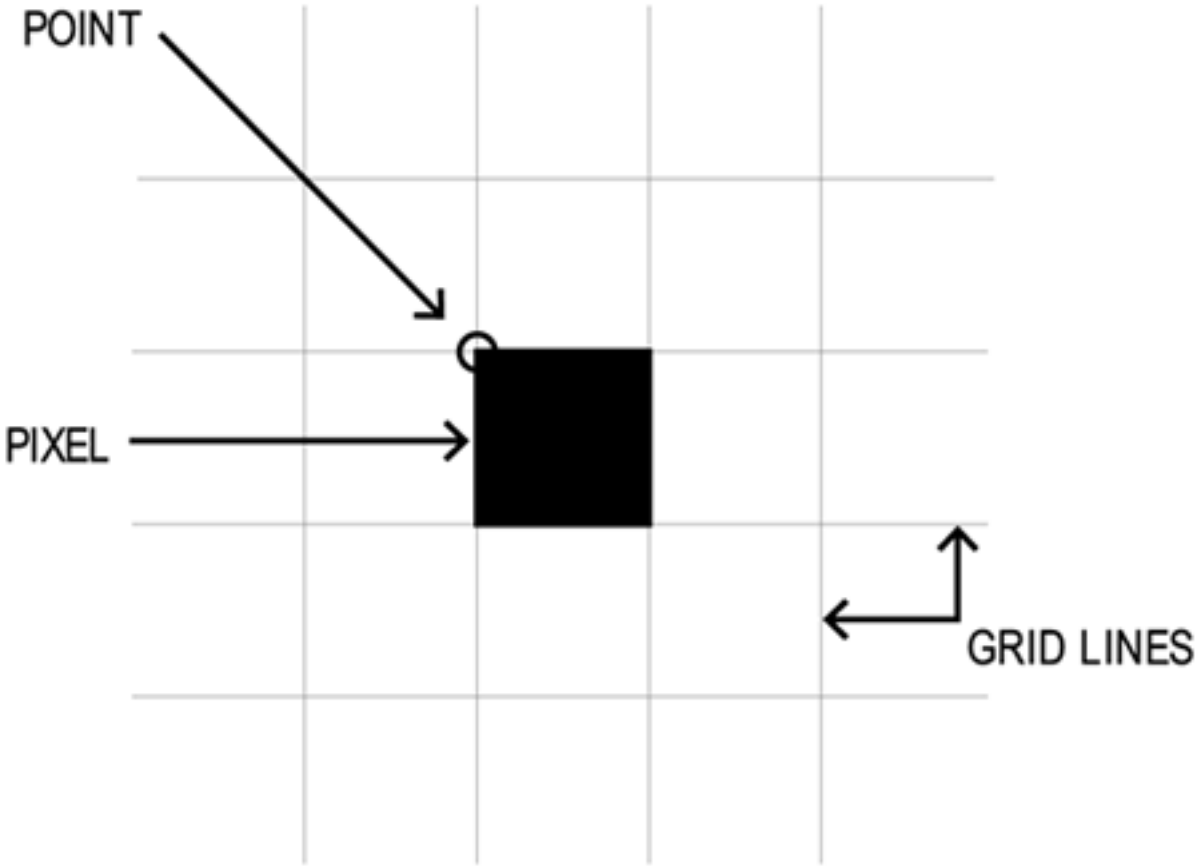


Points

The location on the coordinate plane where imaginary horizontal and vertical grid lines intersect is called a **point.** Points themselves are dimensionless whereas a pixel is not. As

shown in Fig 2, a pixel "hangs" down and to the right of the point by which it is addressed. A pixel thus lies between the infinitely thin lines of the coordinate grid.

Figure 2. A POINT AND A PEXEL



The data type for points is `Point`:

```
struct Point
{
  short v;  // Vertical coordinate.
  short h;  // Horizontal coordinate.
} ;
typedef struct Point Point;
typedef Point *PointPtr;
```

Rectangles

A **rectangle,** whose borders are infinitely thin in the same way that a point is infinitely small, is used to define an area on the screen.

The data type for rectangles is `Rect`:

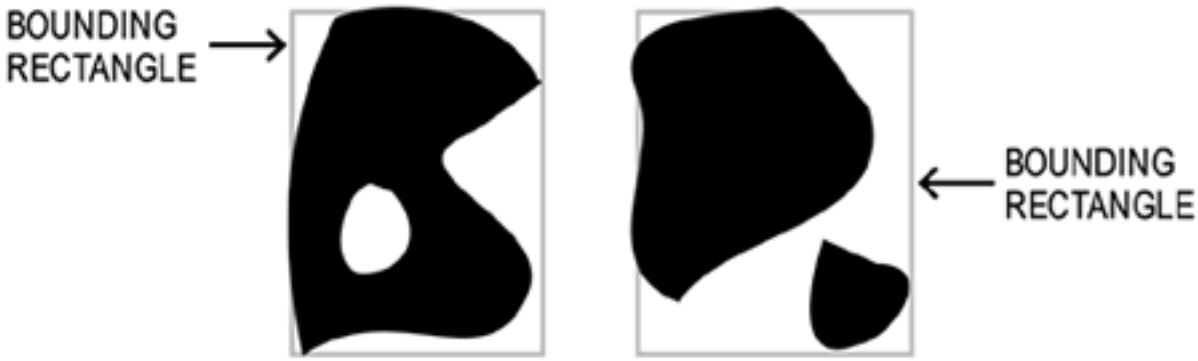
```
struct Rect
{
  short top;
  short left;
  short bottom;
  short right;
} ;
typedef struct Rect Rect;
typedef Rect *RectPtr;
```

If the bottom coordinate of a rectangle is equal to or less than the top, or the right coordinate is less than the left, the rectangle is said to be an **empty rectangle.**

Regions

A **region** is an arbitrary area, or set of areas, the outline of which is one or more closed loops. A region can be concave or convex, can consist of one connected area or many separate ones, and can even have holes in the middle. In the examples in Fig 3, the region on the left has a hole and the one on the right consists of two unconnected areas.

Figure 3. TWO REGIONS



Region Objects and Accessor Functions

QuickDraw stores information about regions in opaque data structures called **region objects.** The data type `RgnHandle` is defined as a reference to a region object:

```
typedef struct OpaqueRgnHandle* RgnHandle;
```

One accessor function is provided to access the information in region objects:

Function	*Description*
`GetRegionBounds`	Get the region's bounding rectangle.

For a region which is a rectangle, the rectangle returned by `GetRegionBounds` defines the entire region. The data for more complex regions is stored in the region object in a proprietary format. The function `IsRegionRectangular` may be used to determine whether a specific region is rectangular.

The Graphics Pen, Foreground and Background Colors, Pixel Patterns and Bit Patterns, and Transfer Modes

The Graphics Pen

The metaphorical graphics pen used for drawing lines and shapes in a graphics port is rectangular in shape and its size (that is, its height and width) is measured in pixels. Whenever you draw into a graphics port, the characteristics of the graphics pen determine how the drawing looks. Those characteristics are as follows:

- Pen **location,** which is specified in local coordinates stored in the graphics port. The functions `Move` and `MoveTo` are used to move the pen to a specified location, and the function `GetPen` gets the pen's current location.
- Pen **size,** which is specified by the a width and a height (in pixels) stored in the graphics port. The pen's default size is one-by-one pixel; however, `PenSize` can be used to change the size and shape up to a 32,767-by-32,767 pixel square. Note that if either the width or height is set to 0, the pen does not draw.
- Pen **color,** that is, the graphics port's foreground color.
- Pen **pattern,** which defines the pattern that the pen draws with.
- Pen **transfer mode,** a Boolean or arithmetic operation which determines how QuickDraw transfers the pen pattern to the pixel map during drawing operations.
- Pen **visibility,** which is specified by an integer stored in the graphics port, indicating whether drawing operations will actually appear. For example, for 0 or negative values, the pen draws with "invisible ink". The functions `ShowPen` and `HidePen` are used to change pen visibility.

Getting and Setting the Pen State

The following functions are used to get and set the current pen state:

Function	*Description*
`GetPenState`	Returns, in a `PenState` structure, the graphics pen's current location, size, transfer mode, and pattern.
`SetPenState`	Using information supplied by a `PenState` structure, sets the graphics pen's location, size, transfer mode, and pattern.
`PenNormal`	Resets the pen size, transfer mode, and pattern to the state initialized when the graphics port was opened.

Foreground and Background Color

Foreground Color

The function `RGBForecolor` is used to set the foreground color in the graphics port. You may also use the Palette Manager function `PmForeColor` to set the foreground color.

The foreground color is used by the graphics pen for **drawing** lines, framed shapes, and text. The foreground color is also used by QuickDraw shape **painting** functions.

Background Color

The function `RGBBackColor` is used to set the background color in the graphics port. You may also use the Palette Manager function `PmBackColor` to set the background color.

The background color is used by QuickDraw **erasing** functions, and is also used by the `ScrollRect` function to replace the scrolled pixels.

Pixel Patterns and Bit Patterns

Pixel Patterns

If you wish to draw or paint with a color *pattern* rather than the color set by `RGBForecolor`, you can set the pen pixel pattern in the graphics port using `SetPortPenPixPat` or `PenPixPat`. (Initially, the pen pixel pattern in the graphics port is all-"black." When you set a non-all-"black" pattern, the pen pattern in the graphics port overrides the foreground color.)

You define a pixel pattern in a `'ppat'` resource. To retrieve the pixel pattern stored in the `'ppat'` resource, you use the `GetPixPat` function. The handle to a `pixPat` data structure returned by `GetPixPat` may then be used in a call to `SetPortPenPixPat` or `PenPixPat` to set the pixel pattern.

Similarly, if you wish to erase with a pixel pattern rather than the background color, or replace the pixels scrolled by `ScrollRect` with a pixel pattern rather than the background color, you can set the background pixel pattern in the graphics port using `SetPortBackPixPat` or `BackPixPat`. (Initially, the background pixel pattern in the graphics port is all-"white." When you assign a non-all-"white" pattern, the background pattern in the graphics port overrides the background color)

In addition to drawing, painting and erasing functions, QuickDraw includes shape **filling** functions, which may be used to fill a specified shape using a specified pixel pattern. A handle to a `pixPat` data structure is passed in the `thePPat` parameter of these functions.

Bit Patterns

After drawing or painting with a pixel pattern, you can return to drawing or painting with the foreground color by simply restoring the default all-"black" pattern by calling `PenPat` and passing in the bit pattern contained in the QuickDraw global variable `black` as follows:

```
Pattern blackPattern;

PenPat(GetQDGlobalsBlack(&blackPattern));
```

After erasing with a pixel pattern, you can return to erasing with the background color by simply restoring the default all-"white" pattern by calling `BackPat` and passing in the bit pattern contained in the QuickDraw global variable `white` as follows:

```
Pattern whitePattern;

BackPat(GetQDGlobalsWhite(&whitePattern));
```

When you use the `PenPat` and `BackPat` functions, QuickDraw constructs a pixel pattern equivalent to the bit pattern, The graphics port's current foreground color is used for the "black" bits in the bit pattern, and the background color is used for the "white" bits.

The `PenPat` and `BackPat` functions may also be used to set other bit patterns in the graphics port.

Transfer Modes

The term **transfer mode** may be considered as a generic term encompassing three different transfer mode types. Each has to do with the way source pixels interact with destination pixels during drawing, painting, erasing, filling, and **copying** operations. The three types of transfer mode are as follows:

- ***Boolean Pattern Mode.*** Boolean pattern modes apply to line drawing, framing, painting, erasing, and filling operations.
- ***Boolean Source Mode.*** Boolean source modes apply to text drawing and copying operations.
- ***Arithmetic Source Mode.*** Arithmetic source modes apply to drawing (including text drawing), painting, and copying operations.

Boolean Pattern Modes

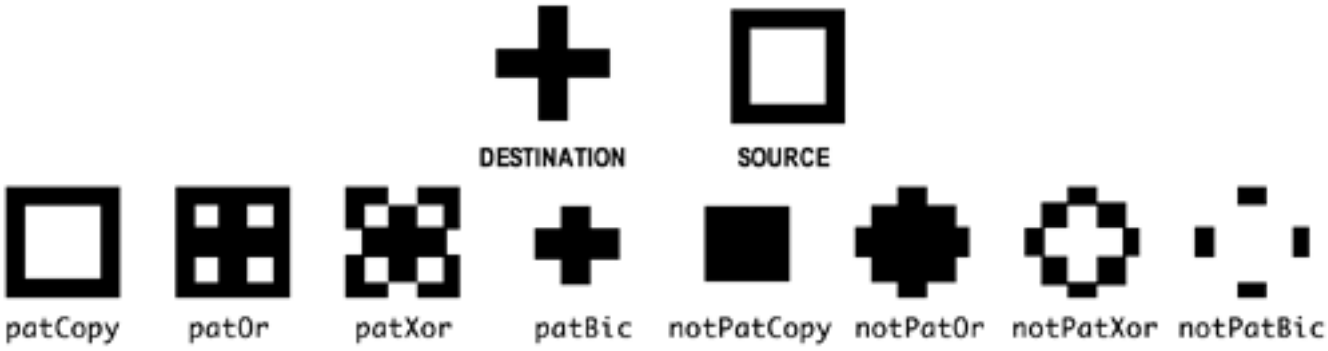
Pattern modes may be set as pen transfer modes in the graphics port using the `PenMode` function. The modes are represented by eight constants, each of which relates to a specific Boolean operation (COPY, OR, XOR, and BIC (for bit clear)) and their inverse variants.

The effects of these modes are best explained assuming a 1-bit (black-and-white) environment in which the foreground color is black and the background color is white. The following lists the pattern modes and describes the effect of source pixels on destination pixels in such an environment.

Pattern Mode	***Action On Destination Pixel***	
	If source pixel is black	*If source pixel is white*
`patCopy`	Apply foreground color	Apply background color
`patOr`	Apply foreground color	Leave alone
`patXor`	Invert	Leave alone
`patBic`	Apply background color	Leave alone
`notPatCopy`	Apply background color	Apply foreground color
`notPatOr`	Leave alone	Force black
`notPatXor`	Leave alone	Invert
`notPatBic`	Leave alone	Apply background color

These effects are illustrated in Fig 4. Note particularly that `patCopy` causes the destination pixels to be completely over-written. `patCopy` is the transfer mode initially set in the graphics port.

Figure 4. EFFECT OF PATTERN MODES



Boolean Source Modes

Boolean source modes may be set as text in the graphics port using the function `TextMode`,

and may be passed as parameters in QuickDraw functions for copying pixel images. The Boolean source modes are the equivalent in text drawing and copying to the Boolean pattern mode used for non-text drawing, painting, filling, and erasing operations.

The relevant constants are `srcCopy`, `srcOr`, `srcXor`, `srcBic`, `notSrcCopy`, `notSrcOr`, `notSrcXor`, and `notSrcBic`. The additional non-standard mode `grayishTextOr` is useful for drawing text in deactivated or disabled user interface objects. (This mode is considered non-standard because it is not stored in pictures and printing with it is undefined.)

Arithmetic Source Modes

Arithmetic source modes may be set in the graphics port, and may be passed as parameters in QuickDraw functions for copying pixel images.

Arithmetic source modes perform arithmetic operations on the values of the red, green and blue components of the source and destination pixels. Because they work with RGB colors rather than color table indexes, arithmetic transfer modes produce predictable results on indexed devices. The arithmetic source modes and their effects in a color environment are as follows:

Constant	*Value*	*Description*
`blend`	`32`	Destination pixel is replaced with a blend of the source and destination pixel colors. Revert to `srcCopy` mode if the destination is a bitmap or 1-bit pixel image.
`addPin`	`33`	Destination pixel is replaced with the sum of the source and destination pixel colors up to a maximum allowable value. Revert to `srcBic` mode if the destination is a bitmap or 1-bit pixel image.
`addOver`	`34`	Destination pixel is replaced with the sum of the source and destination pixel colors, but if the value of the red, green or blue component exceeds 65,536, then subtract 65,536 from that value. Revert to `srcXor` mode if the destination is a bitmap or 1-bit pixel image.
`subPin`	`35`	Destination pixel is replaced with the difference of the source and destination pixel colors, but not less than a minimum allowable value. Revert to `srcOr` mode if the destination is a bitmap or 1-bit pixel image.
`transparent`	`36`	Source and destination pixel are replaced with the source pixel if the source pixel is not equal to the background color.

`addMax`	37	Destination pixel is replaced with the color containing the greater saturation of each of the RGB components of the source and destination pixels. Revert to `srcBic` mode if the destination is a bitmap or 1-bit pixel image.
`subOver`	38	Destination pixel is replaced with the difference of the source and destination pixel colors, but if the value of the red, green or blue is less than 0, add the negative result to 65,536. Revert to `srcXor` mode if the destination is a bitmap or 1-bit pixel image.
`adMin`	39	Destination pixel is replaced with the color containing the lesser saturation of each of the RGB components of the source and destination pixels. Revert to `srcOr` mode if the destination is a bitmap or 1-bit pixel image.

Drawing Lines and Framed Shapes

Functions for Drawing Lines

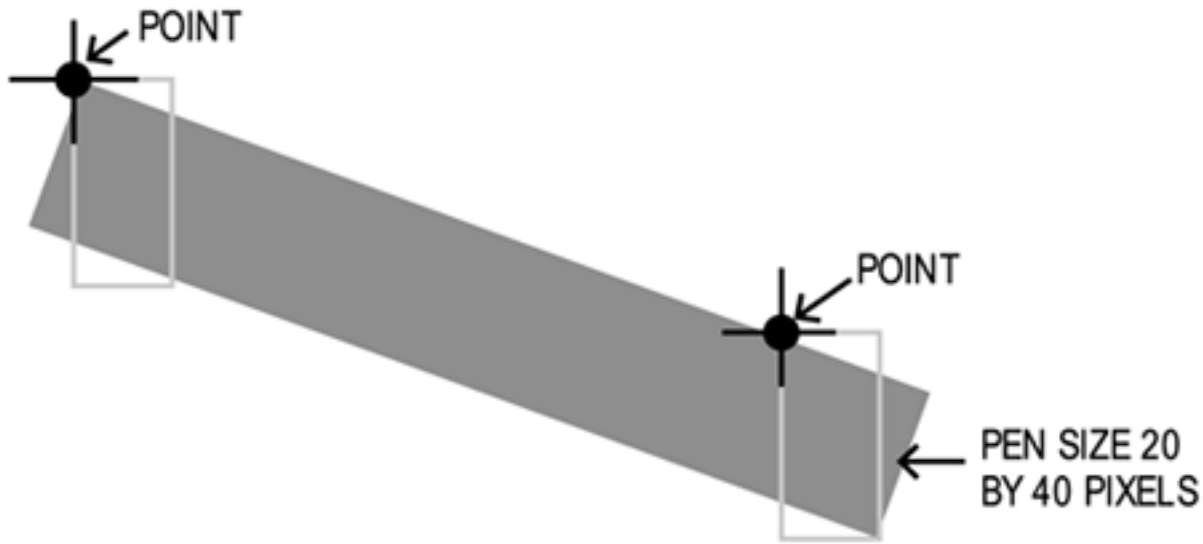
You can move the graphics pen to a specified location, and you can draw lines from that location. Lines are drawn using the current graphics pen size, foreground color or pen pixel/bit pattern, and pen pattern mode.

Functions for moving the graphics pen and drawing lines are as follows:

Function	*Description*
`MoveTo`	Moves the graphics pen location to the specified location, in local coordinates
`Move`	Moves the graphics pen a specified distance from its current location
`LineTo`	Draws a line from the current pen location to the specified location, in local coordinates
`Line`	Draws a line a specified distance from the graphics pen's current location

Fig 5 shows a line drawn with a pen of size 20-by-40 pixels. Note that the pen "hangs" below and to the right of the defining points,

Figure 5. A LINE DRAWN BY Line To OR Line



Functions for Drawing Framed Shapes

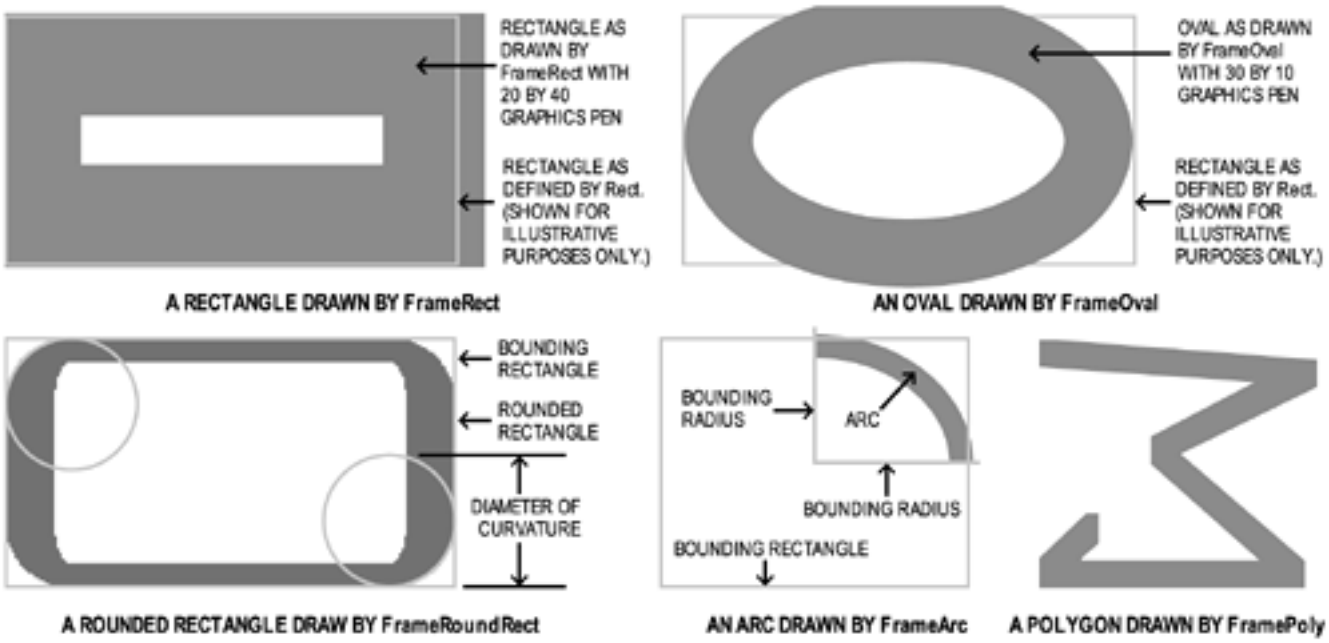
Framing a shape draws its outline only, using the current pen size, foreground color or pen pixel/bit pattern, and pen pattern mode. The pixels in the interior of the shape are unaffected. Framed shapes are drawn using the current graphics pen size, foreground color or pen pixel/bit pattern, and pen pattern mode.

Functions for drawing framed shapes are as follows:

Function	***Description***
`FrameRect`	Draws a rectangle, the position and size of which are defined by a `Rect` structure.
`FrameOval`	Draws an oval, the position and size of which are determined by a bounding rectangle defined by a `Rect` structure.
`FrameRoundRect`	Draws a rounded rectangle, the position and size of which are determined by a bounding rectangle defined by a `Rect` structure. Curvature of the corners is defined by `ovalWidth` and `ovalHeight` parameters.
`FrameArc`	Draws an arc, the position and size of which are determined by a bounding rectangle defined by a `Rect` structure. Starting point and arc extent are determined by `startAngle` and `arcAngle` parameters.
`FramePoly`	Draws a polygon by "playing back" all the line drawing calls that define it.
`FrameRgn`	Draws an outline around a specified region. The line is drawn just inside the region.

Fig 6 shows various framed shapes drawn with various graphics pen sizes and bit patterns. Note that the bounding rectangles completely enclose the shapes they bound, that is, no pixels extend outside the infinitely thin lines of the bounding rectangle. Note also that the arc is a portion of the circumference of an oval bounded by a pair or radii joining at the oval's center.

Figure 6. FRAMED SHAPES DRAWN WITH QUICKDRAW FRAMED SHAPE DRAWING FUNCTIONS


RECTANGLE AS DRAWN BY FrameRect WITH 20 BY 40 GRAPHICS PEN
RECTANGLE AS DEFINED BY Rect. (SHOWN FOR ILLUSTRATIVE PURPOSES ONLY.)
A RECTANGLE DRAWN BY FrameRect
OVAL AS DRAWN BY FrameOval WITH 30 BY 10 GRAPHICS PEN
RECTANGLE AS DEFINED BY Rect. (SHOWN FOR ILLUSTRATIVE PURPOSES ONLY.)
AN OVAL DRAWN BY FrameOval
BOUNDING RECTANGLE
ROUNDED RECTANGLE
DIAMETER OF CURVATURE
A ROUNDED RECTANGLE DRAW BY FrameRoundRect
BOUNDING RADIUS
ARC
BOUNDING RADIUS
BOUNDING RECTANGLE
AN ARC DRAWN BY FrameArc
A POLYGON DRAWN BY FramePoly

Framed Polygons and Regions

Framed polygons and regions require that you call several functions to create and draw them. You begin by calling a function that collects drawing commands into a definition for the object. You then use drawing functions to define the object before calling a function that signals the end of the object definition. Finally, you use a function which draws the newly defined object.

Framed Polygons

To define a polygon you must first call `OpenPoly`. You then call `LineTo` a number of times to create lines from the first vertex to the second, from the second vertex to the third, and so on. You then call `ClosePoly`, which completes the definition process. After defining a polygon in this way, you can draw the polygon, as a framed polygon, using `FramePoly`.

Note that, in the framed polygon in Fig 5, the final defining line from the last vertex back to the first vertex was not drawn during the definition process. Note also that, as in all line drawing, `FramePoly` hangs the pen down and to the right of the infinitely thin lines that define the polygon.

Framed Regions

To define a region, you can use any set of lines or shapes, including other regions, so long as the region's outline consists of one or more closed loops. First, however, you must call `NewRgn` and `OpenRgn`. You then use line, shape, or region drawing commands to define the region. When you have finished collecting commands to define the outline of the region, you call `CloseRgn`. You can then draw the framed region using `FrameRegion`.

Drawing Painted and Filled Shapes

Painting a shape fills both its outline and its interior with the current foreground color or graphics pen pixel/bit pattern. Filling a shape fills both its outline and its interior with a pixel pattern or bit pattern passed in a parameter of the QuickDraw shape filling functions.

Transfer Mode. Painting operations utilize the current graphics pen pattern mode. In filling operations, the transfer mode is invariably the pattern mode `patCopy`, meaning that the destination pixels are always completely overwritten.

Functions for Painting and Filling Shapes

The following lists the available functions for painting and filling shapes:

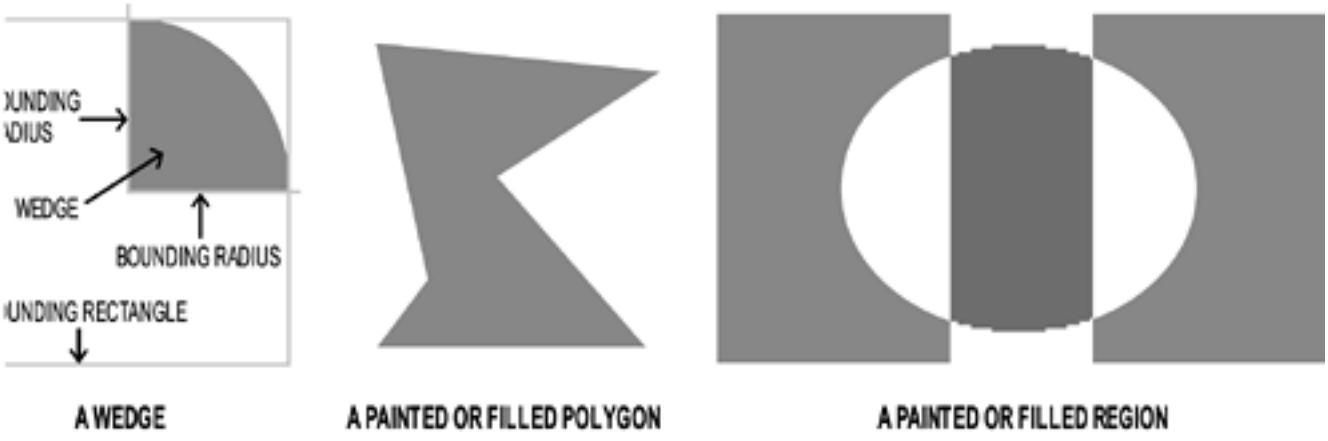
Function	*Description*
`PaintRect`	Fills a rectangle with the current foreground color or graphics pen pixel/ bit pattern.
`PaintOval`	Fills an oval with the current foreground color or graphics pen pixel/ bit pattern.
`PaintRoundRect`	Fills a round rectangle with the current foreground color or graphics pen pixel/ bit pattern.
`PaintArc`	Fills a wedge with the current foreground color or graphics pen pixel/ bit pattern.
`PaintPoly`	Fills a polygon with the current foreground color or graphics pen pixel/ bit pattern.
`PaintRgn`	Fills a region with the current foreground color or graphics pen pixel/ bit pattern.
`FillRect`	Fills a rectangle with a specified bit pattern.
`FillCRect`	Fills a rectangle with a specified pixel pattern.
`FillOval`	Fills an oval with a specified bit pattern.
`FillCOval`	Fills an oval with a specified pixel pattern.
`FillRoundRect`	Fills a round rectangle with a specified bit pattern.
`FillCRoundRect`	Fills a round rectangle with a specified pixel pattern.

`FillArc`	Fills a wedge of an oval with a specified bit pattern.
`FillCArc`	Fills a wedge of an oval with a specified pixel pattern.
`FillPoly`	Fills a polygon with a specified bit pattern.
`FillCPoly`	Fills a polygon with a specified pixel pattern.
`FillRgn`	Fills a region with a specified bit pattern.
`FillCRgn`	Fills a region with a specified pixel pattern.

Wedges

The **wedges** drawn by `PaintArc`, `FillArc`, and `FillCArc` are pie-shaped segments of an oval bounded by a pair of radii joining at the oval's center. A wedge includes part of the oval's interior. Like the framed arc, wedges are defined by the bounding rectangle that encloses the oval, along with a pair of angles marking the positions of the bounding radii. Fig 7 shows a wedge.

Figure 7. A WEDGE, A PAINTED OR FILLED POLYGON, AND A PAINTED OR FILLED REGION



Painted and Filled Polygons and Regions

The general procedure for drawing painted and filled polygons and regions is the same as described for their framed counterparts, above.

Fig 7 shows the polygon as defined for the framed polygon in Fig 6, but this time drawn with one of the polygon painting or filling functions. Note that, although the final defining line from the last vertex back to the first vertex was not drawn, the painting and filling functions complete the polygon (whereas `FramePoly` did not).

Fig 7 also shows a region comprising two rectangles and an overlapping oval, drawn using

`PaintRgn`. Note that, where two regions overlap, the additional area is added to the region and the overlap is removed from the region.

Erasing and Inverting Shapes

Erasing a shape fills both its outline and its interior with the background color or background pixel/bit pattern. Inverting a shape simply inverts all the pixels in the shape; for example, all black pixels become white, and vice versa.

Transfer Mode. In erasing operations, the transfer mode is invariably the pattern mode `patCopy`, meaning that the destination pixels are always completely overwritten.

Functions for Erasing and Inverting Shapes

The following list the available functions for painting and filling shapes:

Function	*Description*
`EraseRect`	Fills a rectangle with the current background color or pixel/ bit pattern.
`EraseOval`	Fills an oval with the current background color or pixel/ bit pattern.
`EraseRoundRect`	Fills a round rectangle with the current background color or pixel/ bit pattern.
`EraseArc`	Fills a wedge with the current background color or pixel/ bit pattern.
`ErasePoly`	Fills a polygon with the current background color or pixel/ bit pattern.
`EraseRgn`	Fills a region with the current background color or pixel/bit pattern.
`InvertRect`	Inverts all the pixels in a rectangle.
`InvertOval`	Inverts all the pixels in an oval.
`InvertRoundRect`	Inverts all the pixels in a round rectangle.
`InvertArc`	Inverts all the pixels in a wedge.
`InvertPoly`	Inverts all the pixels in a polygon.
`InvertRgn`	Inverts all the pixels in a region.

Drawing Pictures

Your application can record a sequence of QuickDraw drawing operations in a **picture** and play its image back later. Fig 8 shows an example of a simple picture containing a filled rectangle, a filled oval, and some text.

Figure 8. A SIMPLE QUICKDRAW PICTURE



The subject of pictures is addressed in more detail in Chapter 13.

Drawing Text

Setting the Font

The font used to draw text in a graphics port may be set using the function `TextFont`. `TextFont` takes a single parameter, of type `SInt16`, which may be either a predefined constant or a **font family ID** number. Although predefined constants remain in the header file Fonts.h, their use is now discouraged by Apple.

You can get the font family ID using `GetFNum`. For example, the following sets the current font to Palatino:

```
SInt16 fontNum;

GetFNum("\pPalatino",&fontNum);
TextFont(fontNum);
```

> **Note**
>
> If you know the font family ID, you can get its name by calling the Font Manager's `GetFontName` function. If you do not know either the font family ID or the font name, you can use the Resource Manager's `GetIndResource` function followed by the `GetResInfo` function to determine the names and IDs of all available fonts.

Setting and Modifying the Text Style

You use the function `TextFace` to change the text style, using any combination of the constants `bold`, `italic`, `underline`, `outline`, `shadow`, `condense`, and `extend`. Some examples are as follows:

```
TextFace(bold);                              // Set to bold.
TextFace(bold | italic);                     // Set to bold and italic.)
TextFace(GetPortTextFace(thePort) | bold);   // Add bold to existing.
TextFace(GetPortTextFace(thePort) &~ bold);  // Remove bold.
TextFace(normal);                            // Set to plain.
```

Setting the Font Size

You use the function `TextSize` to change the font size in typographical **points.** A point is approximately 1/72 inch.

Changing the Width of Characters

Widening and narrowing space and non-space characters lets you meet special formatting requirements.

You use `SpaceExtra` to specify the extra pixels to be added to or subtracted from the standard width of the space character. `SpaceExtra` is ordinarily used in text-justification functions.

Transfer Mode

The transfer mode initially set in the graphics port is the Boolean source mode `srcOr`. This mode causes the color of the glyph to be determined by the foreground color and the drawn glyph to completely overwrite the existing pixels. (In this mode only those bits which make up the actual glyph are drawn.)

> **Note**
>
> A glyph is the visual representation of a character.

You should generally use either `srcOr` or `srcBic` when drawing text, because all other transfer modes draw the character's background as well as the glyph itself. This can result in the clipping of characters by adjacent characters.

Copying Pixel Images Between Graphics Ports

QuickDraw provides the following three functions for copying pixel images between graphics ports:

- `CopyBits`, which copies a pixel image to another graphics port, optionally allowing you to
    - Resize the image
    - Modify the image with transfer modes
    - Clip the image to a region
- `CopyMask`, which copies a pixel image to another graphics port, allowing you to
    - Resize the image
    - Modify the image by passing it through a mask
- `CopyDeepMask`, which combines the effects of `CopyBits` and `CopyMask`, optionally allowing you to
    - Resize the image
    - Clip the image to a region
    - Specify a transfer mode
    - Modify the image by passing it through a mask

The mask used by `CopyMask` and `CopyDeepMask` may be another pixel map whose pixels indicate proportionate weights of the colors for the source and destination pixels.

The `CopyBits`, `CopyMask`, and `CopyDeepMask` functions date from the era of black-and-white Macintoshes, which is why they expect a pointer to a bitmap in their source and destination parameters. Thus, when you are copying pixel maps using these functions, you must cast the address of the handle to the pixel map to a pointer to a bitmap. By looking at certain information in the graphics port object, `CopyBits`, `CopyMask`, and `CopyDeepMask` can establish that you have passed the functions a handle to a pixel map

rather than the base address of a bitmap.

Using Masks

With `CopyMask` and `CopyDeepMask`, you supply a pixel map to act as the mask. The mask's pixels proportionally select between source and destination pixel values.

In the case of masks that are 1 bit deep, black bits in the mask cause the destination pixel to take the color of the source pixel and white bits cause the destination pixel to retain its current color. In the case of masks with pixel depths greater than 1, Color QuickDraw takes a weighted average between source and destination colors. For example, a blue mask (that is, one with high values for the blue components of all pixels) filters out blue values coming from the source.

Transfer Modes

`CopyBits` and `CopyDeepMask` both allow you to specify the transfer mode, which can be either a Boolean source mode or an arithmetic source mode.

The Importance of Foreground and Background Color

Applying a foreground color other than black or a background color other than white to the pixel can produce an unexpected result. For consistent results, you should set the foreground color to black and the background color to white before using `CopyBits`, `CopyMask`, or `CopyDeepMask`. (That said, setting foreground and background colors to something other than black or white can achieve some interesting coloration effects.)

Dithering

You can use **dithering** with `CopyBits` and `CopyDeepMask`. Dithering is a technique involving the mixing of existing colors to create the illusion of a third color. This is most useful for images displayed on indexed devices, which can only display a limited number of colors at any one time.

You can add dithering to any transfer mode by adding the following constant to the transfer mode:

```
ditherCopy = 64    // Add to source mode for dithering.
```

Copying from Offscreen Graphics Ports

To gracefully display complex images, your application should construct the image in an **offscreen graphics world** and then use `CopyBits` to transfer the image to the onscreen

graphics port. (Offscreen graphics worlds are addressed in Chapter 13.)

Scrolling Pixels in the Port Rectangle

You can use `ScrollRect` to scroll the pixels in the port rectangle. `ScrollRect` takes four parameters: the rectangle to scroll, a horizontal distance to scroll, a vertical distance to scroll, and a region reference. `ScrollRect` is a special form of `CopyBits` which copies bits enclosed by a rectangle and stores them within that same rectangle. The vacated area is filled with the current background color or pixel/bit pattern.

Manipulating Rectangles and Regions

QuickDraw provides many functions for manipulating rectangles and regions. You can use the functions that manipulate rectangles to manipulate any shape based on a rectangle, that is, rounded rectangles, ovals, arcs, and wedges.

For example, you could define a rectangle to bound an oval and then frame the oval. You could then use `OffsetRect` to move the oval's bounding rectangle down. Using the offset bounding rectangle, you could frame a second, connected oval to form a figure eight with the first oval. You could then use that shape to help define a region. You could create a second region, and then use `UnionRgn` to create a region from the union of the two.

Manipulating Rectangles

The following summarizes the functions for manipulating and performing calculations on rectangles:

Function	*Description*
`EmptyRect`	Determines whether a rectangle is an empty rectangle
`EqualRect`	Determines whether two rectangles are equal
`InsetRect`	Shrinks or expands a rectangle
`OffsetRect`	Moves a rectangle
`PtInRect`	Determines whether a pixel is enclosed in a rectangle
`PtToAngle`	Calculates the angle from the middle of a rectangle to a point
`Pt2Rect`	Determines the smallest rectangle that encloses two points
`SectRect`	Determines whether two rectangles intersect
`UnionRect`	Calculates the smallest rectangle that encloses two rectangles

Manipulating Regions

The following summarizes the functions for manipulating, and performing calculations on, regions:

Function	*Description*

`CopyRgn`	Makes a copy of a region
`DiffRgn`	Subtracts one region from another
`EmptyRgn`	Determines whether a region is empty
`EqualRgn`	Determines whether two regions have identical sizes, shapes, and locations
`InsetRgn`	Shrinks or expands a region
`OffsetRgn`	Moves a region
`PtInRgn`	Determines whether a pixel is within a region
`RectInRgn`	Determines whether a rectangle intersects a region
`RectRgn`	Changes the structure of an existing region to that of a rectangle (using a `Rect`)
`SectRgn`	Calculates the intersection of two regions
`SetEmptyRgn`	Sets a region to empty
`SetRectRgn`	Changes the structure of an existing region to that of a rectangle (using coordinates)
`UnionRgn`	Calculates the union of two regions
`XorRgn`	Calculates the difference between the union and the intersection of two regions

Manipulating Polygons

You can use `OffSetPoly` to move a polygon; however, QuickDraw provides no other functions for manipulating polygons.

Scaling Shapes and Regions Within the Same Graphics Port

To scale shapes and regions within the same graphics port, you can use the functions `ScalePt`, `MapPt`, `MapRect`, `MapRgn`, and `MapPoly`.

Highlighting

Highlighting is used when selecting and deselecting objects such as text or graphics. TextEdit, for example, uses highlighting to indicate selected text. If the current highlight color is, for example, blue, TextEdit draws the selected text, then uses `InvertRgn` to produce a blue background for the text.

The **system highlight color,** which can be changed by the user at the Highlight Color item in the Appearance pane of the Appearance control panel, is stored in a low memory global represented by the symbolic name `HiliteRGB`. It can be retrieved using `LMGetHiliteRGB`. You can override the default color using the function `HiliteColor`. The current color is copied to the graphics port object, and may be retrieved from there using the function `GetPortHiliteColor`.

Color QuickDraw implements highlighting by replacing the background color with the highlight color. Another low memory global, represented by the symbolic name `HiliteMode`, contains a byte that represents the current highlight mode. One bit in that byte, represented by the constant `pHiliteBit`, is used to toggle the background and highlight colors.

Because Color QuickDraw resets the highlight bit after performing each drawing operation, your application must always clear the highlight bit immediately before calling `InvertRgn` (or, indeed, any of the other drawing or image-copying function that uses the `patXor` or `srcXor` transfer modes.)

The highlight mode can be retrieved and set using `LMGetHiliteMode` and `LMSetHiliteMode`, and `BitClr` may be used to clear the highlight bit:

```
UInt8 hiliteMode;
...
hiliteMode = LMGetHiliteMode();
BitClr(&hiliteMode,pHiliteBit);
LMSetHiliteMode(hiliteMode);
```

Another way to use highlighting is to add this constant to the transfer mode you pass in calls to the functions `PenMode`, `CopyBits`, `CopyDeepMask` and `TextMode`:

```
hilite = 50  // Add to source or pattern mode for highlighting.
```

Drawing Other Graphics Entities

In addition to drawing lines, rectangles, rounded rectangles, ovals, arcs, wedges, polygons and regions, and text, you can also use QuickDraw to draw the following:

- Cursors
- Icons

Cursors and Icons are addressed in Chapter 13.

Saving and Restoring the Graphics Port Drawing State

As stated above, the functions `GetPenState` and `SetPenState` are used to save and restore the graphics pen's location, size, transfer mode, and pattern, and `PenNormal` is used to initialize the pen's size, transfer mode, and pattern.

Typically, an application calls `GetPenState` at the beginning of a function that changes the pen's location, size, transfer mode, and/or pattern and restores the saved state to the pen on exit from that function. Depending on its requirements, an application might also save and restore the graphics port's foreground and background colors, and the text transfer mode, in the same way.

Since the introduction of the Appearance Manager, it has also become necessary to save and restore the pen pixel/bit pattern and background pixel/bit pattern in functions that call the Appearance Manager functions `SetThemeBackground`, `SetThemePen`, and/or `SetThemeWindowBackground`. Recall from Chapter 6: The Appearance Manager that constants of type `ThemeBrush` are passed in the `inBrush` parameter of these Appearance Manager functions and that the value in the `inBrush` parameter can represent either a color or a pattern depending on the current appearance.

Accordingly, in the era of the Appearance Manager, applications that call `SetThemeBackground` and/or `SetThemePen` will need to take measures to save and restore the *complete* graphics port drawing state and, if required, normalize that state.

The following functions are used for saving, restoring, and normalizing the graphics port drawing state:

Function	***Description***
`GetThemeDrawingState`	Obtains the drawing state of the current graphics port.
`SetThemeDrawingState`	Sets the drawing state of the current graphics port.
`NormalizeThemeDrawingState`	Sets the current graphics port to the default drawing state.

`DisposeThemeDrawingState`	Releases the memory associated with a reference to a graphics port's drawing state. (Note that this memory may also be released by passing true in the `inDisposeNow` parameter of `SetThemeDrawingState`.)

Information about the current state of the graphics port is stored in a structure of type `ThemeDrawingState`. This is a private data structure.

Main QuickDraw Constants, Data Types and Functions

Constants

Boolean Pattern Modes

```
patCopy      = 8
patOr        = 9
patXor       = 10
patBic       = 11
notPatCopy   = 12
notPatOr     = 13
notPatXor    = 14
notPatBic    = 15
```

Boolean Source Modes

```
srcCopy      = 0
srcOr        = 1
srcXor       = 2
srcBic       = 3
notSrcCopy   = 4
notSrcOr     = 5
notSrcXor    = 6
notSrcBic    = 7
```

Arithmetic Transfer Modes

```
blend        = 32
addPin       = 33
addOver      = 34
subPin       = 35
transparent  = 36
addMax       = 37
subOver      = 38
adMin        = 39
```

Add Dithering to Transfer Modes

```
ditherCopy   = 64
```

Highlighting

```
hilite       = 50
hiliteBit    = 7
pHiliteBit   = 0
```

Pattern List Resource ID for Pattern Resources in the System File

```
sysPatListID = 0
```

Data Types

Point

```
struct Point
{
```

```
  short   v;
  short   h;
} ;

typedef struct Point Point;
typedef Point *PointPtr;
```

Rect

```
struct Rect
{
  short   top;
  short   left;
  short   bottom;
  short   right;
} ;

typedef struct Rect Rect;
typedef Rect *RectPtr;
```

Region

```
typedef struct OpaqueRgnHandle *RgnHandle;
```

Polygon

```
struct Polygon
{
  short   polySize;
  Rect    polyBBox;
  Point   polyPoints[1];
} ;

typedef struct Polygon Polygon;
typedef Polygon *PolyPtr, **PolyHandle;
```

PenState

```
struct PenState
{
  Point   pnLoc;
  Point   pnSize;
  short   pnMode;
  Pattern pnPat;
} ;

typedef struct PenState PenState;
```

Functions

Managing the Graphics Pen

```
void          HidePen(void);
void          ShowPen(void);
void          GetPen(Point *pt);
void          GetPenState(PenState *pnState);
void          SetPenState(const PenState *pnState);
void          PenSize(short width,short height);
void          PenMode(short mode);
void          PenNormal(void);
```

Getting and Setting Foreground, Background, and Pixel Color

```
void          RGBForeColor(const RGBColor *color);
void          RGBBackColor(const RGBColor *color);
void          GetForeColor(RGBColor *color);
void          GetBackColor(RGBColor *color);
void          GetCPixel(short h,short v,RGBColor *cPix);
void          SetCPixel(short h,short v,const RGBColor *cPix);
```

Creating and Disposing of Pixel Patterns

```
PixPatHandle  GetPixPat(short patID);
PixPatHandle  NewPixPat(void);
void          CopyPixPat(PixPatHandle srcPP,PixPatHandle dstPP);
void          MakeRGBPat(PixPatHandle pp,const RGBColor *myColor);
void          DisposePixPat(PixPatHandle pp);
```

Getting Pattern Resources

```
PatHandle     GetPattern(short patternID);
void          GetIndPattern(Pattern *thePat,short patternListID,short index);
```

Changing the Pen and BackGround Pixel Pattern and Bit Pattern

```
void          BackPixPat(PixPatHandle pp);
void          PenPixPat(PixPatHandle pp);
void          BackPat(const Pattern *pat);
void          PenPat(const Pattern *pat);
```

Drawing Lines

```
void          MoveTo(short h,short v);
void          Move(short dh,short dv);
void          LineTo(short h,short v);
void          Line(short dh,short dv);
```

Drawing Rectangles

```
void          FrameRect(const Rect *r);
void          PaintRect(const Rect *r);
void          FillRect(const Rect *r,ConstPatternParam pat);
void          FillCRect(const Rect *r,PixPatHandle pp);
void          InvertRect(const Rect *r);
void          EraseRect(const Rect *r);
```

Drawing Rounded Rectangles

```
void          FrameRoundRect(const Rect *r,short ovalWidth,short ovalHeight);
void          PaintRoundRect(const Rect *r,short ovalWidth,short ovalHeight);
void          FillRoundRect(const Rect *r,short ovalWidth,short ovalHeight,const Pattern
*pat);
void          FillCRoundRect(const Rect *r,short ovalWidth,short ovalHeight,PixPatHandle
pp);
void          InvertRoundRect(const Rect *r,short ovalWidth,short ovalHeight);
void          EraseRoundRect(const Rect *r,short ovalWidth,short ovalHeight);
```

Drawing Ovals

```
void          FrameOval(const Rect *r);
void          PaintOval(const Rect *r);
void          FillOval(const Rect *r,const Pattern *pat);
void          FillCOval(const Rect *r,PixPatHandle pp);
```

```
void          InvertOval(const Rect *r);
void          EraseOval(const Rect *r);
```

Drawing Arcs and Wedges

```
void          FrameArc(const Rect *r,short startAngle,short arcAngle);
void          PaintArc(const Rect *r,short startAngle,short arcAngle);
void          FillArc(const Rect *r,short startAngle,short arcAngle,const Pattern *pat);
void          FillCArc(const Rect *r,short startAngle,short arcAngle,PixPatHandle pp);
void          InvertArc(const Rect *r,short startAngle,short arcAngle);
void          EraseArc(const Rect *r,short startAngle,short arcAngle);
```

Drawing and Painting Polygons

```
void          FramePoly(PolyHandle poly);
void          PaintPoly(PolyHandle poly);
void          FillPoly(PolyHandle poly,const Pattern *pat);
void          FillCPoly(PolyHandle poly,PixPatHandle pp);
void          InvertPoly(PolyHandle poly);
void          ErasePoly(PolyHandle poly);
```

Drawing Regions

```
void          FrameRgn(RgnHandle rgn);
void          PaintRgn(RgnHandle rgn);
void          FillCRgn(RgnHandle rgn,PixPatHandle pp);
void          EraseRgn(RgnHandle rgn);
void          InvertRgn(RgnHandle rgn);
void          FillRgn(RgnHandle rgn, const Pattern *pat);
```

Setting Text Characteristics

```
void          TextFont(short font);
void          TextFace(short face);
void          TextMode(short mode);
void          TextSize(short size);
void          SpaceExtra(Fixed extra);
void          GetFontInfo(FontInfo *info);
```

Drawing and Measuring Text

```
void          DrawChar(short ch);
void          DrawString(ConstStr255Param s);
void          DrawText(const void *textBuf,short firstByte,short byteCount);
short         CharWidth(short ch);
short         StringWidth(ConstStr255Param s);
```

Copying Images

```
void          CopyBits(const BitMap *srcBits,const BitMap *dstBits,const Rect *srcRect,
              const Rect *dstRect,short mode,RgnHandle maskRgn);
void          CopyMask(const BitMap *srcBits,const BitMap *maskBits,const BitMap *dstBits,
              const Rect *srcRect,const Rect *maskRect,const Rect *dstRect);
void          CopyDeepMask(const BitMap *srcBits,const BitMap *maskBits,const BitMap
*dstBits,
              const Rect *srcRect,const Rect *maskRect,const Rect *dstRect,short mode,
              RgnHandle maskRgn);
```

Getting and Setting the Highlight Color and HighLight Mode

```
void          HiliteColor(const RGBColor *color);
```

```
void          LMGetHiliteRGB(RGBColor *hiliteRGBValue);
void          LMSetHiliteRGB(const RGBColor *hiliteRGBValue);
UInt8         LMGetHiliteMode(void);
void          LMSetHiliteMode(UInt8 value);
```

Creating and Disposing of Color Tables

```
CtabHandle    GetCTable(short ctID);
void          DisposeCTable(CTabHandle cTable);
```

Creating and Managing Polygons

```
PolyHandle    OpenPoly(void);
void          ClosePoly(void);
void          KillPoly(PolyHandle poly);
void          OffsetPoly(PolyHandle poly,short dh,short dv);
```

Creating and Managing Rectangles

```
void          SetRect(Rect *r,short left,short top,short right,short bottom);
void          OffsetRect(Rect *r,short dh,short dv);
void          InsetRect(Rect *r,short dh,short dv);
Boolean       SectRect(const Rect *src1,const Rect *src2,Rect *dstRect);
void          UnionRect(const Rect *src1,const Rect *src2,Rect *dstRect);
Boolean       PtInRect(Point pt,const Rect *r);
void          Pt2Rect(Point pt1,Point pt2,Rect *dstRect);
void          PtToAngle(const Rect *r,Point pt,short *angle);
Boolean       EqualRect(const Rect *rect1,const Rect *rect2);
Boolean       EmptyRect(const Rect *r);
```

Creating and Managing Regions

```
RgnHandle     NewRgn(void);
void          OpenRgn(void);
void          CloseRgn(RgnHandle dstRgn);
void          DisposeRgn(RgnHandle rgn);
void          CopyRgn(RgnHandle srcRgn,RgnHandle dstRgn);
void          SetEmptyRgn(RgnHandle rgn);
void          SetRectRgn(RgnHandle rgn,short left,short top,short right,short bottom);
void          RectRgn(RgnHandle rgn,const Rect *r);
void          OffsetRgn(RgnHandle rgn,short dh,short dv);
void          InsetRgn(RgnHandle rgn,short dh,short dv);
void          SectRgn(RgnHandle srcRgnA,RgnHandle srcRgnB,RgnHandle dstRgn);
void          UnionRgn(RgnHandle srcRgnA,RgnHandle srcRgnB,RgnHandle dstRgn);
void          DiffRgn(RgnHandle srcRgnA,RgnHandle srcRgnB,RgnHandle dstRgn);
void          XorRgn(RgnHandle srcRgnA,RgnHandle srcRgnB,RgnHandle dstRgn);
Boolean       PtInRgn(Point pt,RgnHandle rgn);
Boolean       RectInRgn(const Rect *r,RgnHandle rgn);
Boolean       EqualRgn(RgnHandle rgnA,RgnHandle rgnB);
Boolean       EmptyRgn(RgnHandle rgn);
OSErr         BitMapToRegion(RgnHandle region,const BitMap *bMap);
```

Scaling and Mapping Points, Rectangles, Polygons, and Regions

```
void          ScalePt(Point *pt,const Rect *srcRect,const Rect *dstRect);
void          MapPt(Point *pt,const Rect *srcRect,const Rect *dstRect);
void          MapRect(Rect *r,const Rect *srcRect,const Rect *dstRect);
void          MapRgn(RgnHandle rgn,const Rect *srcRect,const Rect *dstRect);
void          MapPoly(PolyHandle poly,const Rect *srcRect,const Rect *dstRect);
```

Determining Whether QuickDraw Has Finished Drawing

```
Boolean       QDDone(GrafPtr port);
```

Retrieving Color QuickDraw Result Codes

```
short         QDError(void);
```

Managing Port Rectangles and Clipping Regions

```
void          ScrollRect(const Rect *r,short dh,short dv,RgnHandle updateRgn);
void          SetOrigin(short h,short v);
void          PortSize(short width,short height);
void          MovePortTo(short leftGlobal,short topGlobal);
void          GetClip(RgnHandle rgn);
void          SetClip(RgnHandle rgn);
void          ClipRect(const Rect *r);
```

Manipulating Points in Graphics Ports

```
void          GlobalToLocal(Point *pt);
void          LocalToGlobal(Point *pt);
void          AddPt(Point src,Point *dst);
void          SubPt(Point *src,Point *dst);
void          SetPt(Point *pt,short h,short v);
Boolean       EqualPt(Point pt1,Point pt2);
Boolean       GetPixel(short h,short v);
```

Relevant Appearance Manager Data Types and Functions

Data Types

```
typedef struct OpaqueThemeDrawingState *ThemeDrawingState;
```

Functions

```
OSStatus      NormalizeThemeDrawingState(void);
OSStatus      GetThemeDrawingState(ThemeDrawingState *outState);
OSStatus      SetThemeDrawingState(ThemeDrawingState inState,Boolean inDisposeNow);
OSStatus      DisposeThemeDrawingState(ThemeDrawingState inState);
```

Demonstration Program QuickDraw Listing

```
//
**********************************************************************************************
// QuickDraw.c                                                          CLASSIC EVENT
MODEL
//
**********************************************************************************************
//
// This program opens a window in which the results of various QuickDraw drawing operations
// are displayed. Individual line and text drawing, framing, painting, filling, erasing,
// inverting, copying, etc., operations are chosen from a Demonstration pull-down menu.
//
// To keep the non-QuickDraw code to a minimum, the program contains no functions for
// updating the window or for responding to activate and operating system events.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  'WIND' resources for the main window, and a small window used for the CopyBits
//    demonstration (purgeable) (initially visible).
//
// •  An 'MBAR' resource and associated 'MENU' resources (preload, non-purgeable).
//
// •  Two 'ICON' resources (purgeable) used for the boolean source modes demonstration.
//
// •  Two 'PICT' resources (purgeable) used in the arithmetic source modes demonstration.
//
// •  'STR#' resources (purgeable) containing strings used in the source modes and text
//    drawing demonstrations.
//
// •  Three 'ppat' resources (purgeable), two of which are used in various drawing,
//    framing, painting, filling, and erasing demonstrations. The third is used in the
//    drawing with mouse demonstration.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//

//
**********************************************************************************************

//  ..........................................................................
includes

#include <Carbon.h>

// ..........................................................................
defines

#define rMenubar                128
#define mAppleApplication       128
#define  iAbout                 1
#define mFile                   129
#define  iQuit                  12
#define mDemonstration          131
#define  iLine                  1
#define  iFrameAndPaint         2
#define  iFillEraseInvert       3
#define  iPolygonRegion         4
#define  iText                  5
```

```
#define  iScrolling            6
#define  iBooleanSourceModes   7
#define  iArithmeticSourceModes 8
#define  iHighlighting         9
#define  iDrawWithMouse        10
#define  iDrawingState         11
#define rWindow                128
#define rPixelPattern1         128
#define rPixelPattern2         129
#define rPixelPattern3         130
#define rDestinationIcon       128
#define rSourceIcon            129
#define rFontsStringList       128
#define rBooleanStringList     129
#define rArithmeticStringList  130
#define rPicture               128
#define MAX_UINT32             0xFFFFFFFF

// ..................................................................... global
variables

Boolean   gRunningOnX     = false;
Boolean   gDone;
WindowRef gWindowRef;
Boolean   gDrawWithMouseActivated;
SInt16    gPixelDepth;
Boolean   gIsColorDevice = false;
RGBColor  gWhiteColor    = {  0xFFFF, 0xFFFF, 0xFFFF } ;
RGBColor  gBlackColor    = {  0x0000, 0x0000, 0x0000 } ;
RGBColor  gRedColor      = {  0xAAAA, 0x0000, 0x0000 } ;
RGBColor  gYellowColor   = {  0xFFFF, 0xCCCC, 0x0000 } ;
RGBColor  gGreenColor    = {  0x0000, 0x9999, 0x0000 } ;
RGBColor  gBlueColor     = {  0x6666, 0x6666, 0x9999 } ;

// ..................................................................... function
prototypes

void    main                   (void);
void    doPreliminaries        (void);
OSErr   quitAppEventHandler    (AppleEvent *,AppleEvent *,SInt32);
void    doEvents               (EventRecord *);
void    doDemonstrationMenu    (MenuItemIndex);
void    doLines                (void);
void    doFrameAndPaint        (void);
void    doFillEraseInvert      (void);
void    doPolygonAndRegion     (void);
void    doScrolling            (void);
void    doText                 (void);
void    doBooleanSourceModes   (void);
void    doArithmeticSourceModes (void);
void    doHighlighting         (void);
void    doDrawWithMouse        (void);
void    doDrawingState         (void);
void    doDrawingStateProof    (SInt16);
void    doGetDepthAndDevice    (void);
UInt16  doRandomNumber         (UInt16,UInt16);
// ****************************************************************************
main

void  main(void)
{
  UInt32        seconds;
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  EventRecord   eventStructure;
```

```
  Boolean       gotEvent;

  // ..................................................................... do
prelimiaries

  doPreliminaries();

  // ..................................................... seed random number
generator

  GetDateTime(&seconds);
  SetQDGlobalsRandomSeed(seconds);

  // ..................................................... set up menu bar and
menus

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    ExitToShell();
  SetMenuBar(menubarHdl);
  DrawMenuBar();

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
      DisableMenuItem(menuRef,0);
    }

    gRunningOnX = true;
  }

  // ..................................................................... open
window

  if(!(gWindowRef = GetNewCWindow(rWindow,NULL,(WindowRef)-1)))
    ExitToShell();

  SetPortWindowPort(gWindowRef);
  UseThemeFont(kThemeSmallSystemFont,smSystemScript);

  // ....... get pixel depth and whether color device for certain Appearance Manager
functions

  doGetDepthAndDevice();
  // .....................................................................
eventLoop

  gDone = false;

  while(!gDone)
  {
    gotEvent = WaitNextEvent(everyEvent,&eventStructure,MAX_UINT32,NULL);
    if(gotEvent)
      doEvents(&eventStructure);
  }
}

// ************************************************************************
doPreliminaries

void  doPreliminaries(void)
```

```
{
  OSErr osError;

  MoreMasterPointers(32);
  InitCursor();
  FlushEvents(everyEvent,0);

  osError = AEInstallEventHandler(kCoreEventClass,kAEQuitApplication,
                            NewAEEventHandlerUPP((AEEventHandlerProcPtr)
quitAppEventHandler),
                            0L,false);
  if(osError != noErr)
    ExitToShell();
}

// ********************************************************************************
doQuitAppEvent

OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  OSErr    osError;
  DescType returnedType;
  Size     actualSize;

  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,
NULL,0,
                              &actualSize);

  if(osError == errAEDescNotFound)
  {
    gDone = true;
    osError = noErr;
  }
  else if(osError == noErr)
    osError = errAEParamMissed;

  return osError;
}
// ********************************************************************************
doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
  SInt32          menuChoice;
  MenuID          menuID;
  MenuItemIndex   menuItem;
  WindowPartCode  partCode;
  WindowRef       windowRef;

  switch(eventStrucPtr->what)
  {
    case kHighLevelEvent:
      AEProcessAppleEvent(eventStrucPtr);
      break;

    case keyDown:
      if((eventStrucPtr->modifiers & cmdKey) != 0)
      {
        menuChoice = MenuEvent(eventStrucPtr);
        menuID = HiWord(menuChoice);
        menuItem = LoWord(menuChoice);
        if(menuID == mFile && menuItem  == iQuit)
          gDone = true;
      }
      break;
```

```
    case mouseDown:
      if(partCode = FindWindow(eventStrucPtr->where,&windowRef))
      {
        switch(partCode)
        {
          case inMenuBar:
            menuChoice = MenuSelect(eventStrucPtr->where);
            menuID = HiWord(menuChoice);
            menuItem = LoWord(menuChoice);

            if(menuID == 0)
              return;

            switch(menuID)
            {
              case mAppleApplication:
                if(menuItem == iAbout)
                  SysBeep(10);
                break;

              case mFile:
                if(menuItem == iQuit)
                  gDone = true;
                break;
              case mDemonstration:
                doDemonstrationMenu(menuItem);
                break;
            }
            break;

          case inDrag:
            DragWindow(windowRef,eventStrucPtr->where,NULL);
            break;

          case inContent:
            if(windowRef != FrontWindow())
              SelectWindow(windowRef);
            else
              if(gDrawWithMouseActivated)
                doDrawWithMouse();
            break;
        }
      }
      break;

    case updateEvt:
      windowRef = (WindowRef) eventStrucPtr->message;
      BeginUpdate(windowRef);
      EndUpdate(windowRef);
      break;
  }
}

// *********************************************************************
doDemonstrationMenu

void  doDemonstrationMenu(MenuItemIndex menuItem)
{
  Rect    portRect;
  Pattern whitePattern;

  gDrawWithMouseActivated = false;

  switch(menuItem)
  {
    case iLine:
```

```
      doLines();
      break;

    case iFrameAndPaint:
      doFrameAndPaint();
      break;

    case iFillEraseInvert:
      doFillEraseInvert();
      break;

    case iPolygonRegion:
      doPolygonAndRegion();
      break;
    case iText:
      doText();
      break;

    case iScrolling:
      doScrolling();
      break;

    case iBooleanSourceModes:
      doBooleanSourceModes();
      break;

    case iArithmeticSourceModes:
      doArithmeticSourceModes();
      break;

    case iHighlighting:
      doHighlighting();
      break;

    case iDrawWithMouse:
      SetWTitle(gWindowRef,"\pDrawing with the mouse");
      RGBBackColor(&gWhiteColor);
      GetWindowPortBounds(gWindowRef,&portRect);
      FillRect(&portRect,GetQDGlobalsWhite(&whitePattern));
      gDrawWithMouseActivated = true;
      break;

    case iDrawingState:
      doDrawingState();
      break;
  }

  HiliteMenu(0);
}

// ********************************************************************************
doLines

void  doLines(void)
{
  Rect         portRect, newClipRect;
  Pattern      whitePattern, systemPattern, blackPattern;
  RgnHandle    oldClipRgn;
  SInt16       a, b, c;
  RGBColor     theColor;
  UInt32       finalTicks;
  PixPatHandle pixpatHdl;

  PenNormal();

  RGBBackColor(&gBlueColor);
```

```
  GetWindowPortBounds(gWindowRef,&portRect);
  FillRect(&portRect,GetQDGlobalsWhite(&whitePattern));
  newClipRect = portRect;
  InsetRect(&newClipRect,10,10);
  oldClipRgn = NewRgn();
  GetClip(oldClipRgn);
  ClipRect(&newClipRect);

  // ............................... lines drawn with foreground color and black pen
pattern

  SetWTitle(gWindowRef,"\pDrawing lines with colors");
  RGBBackColor(&gWhiteColor);
  FillRect(&portRect,&whitePattern);

  if(!gRunningOnX)
    SetThemeCursor(kThemeWatchCursor);

  for(a=1;a<60;a++)
  {
    b = doRandomNumber(0,portRect.right - portRect.left);
    c = doRandomNumber(0,portRect.right - portRect.left);

    theColor.red   = doRandomNumber(0,65535);
    theColor.green = doRandomNumber(0,65535);
    theColor.blue  = doRandomNumber(0,65535);
    RGBForeColor(&theColor);

    PenSize(a * 2,1);

    MoveTo(b,portRect.top);
    LineTo(c,portRect.bottom);

    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
    Delay(2,&finalTicks);
  }

  if(!gRunningOnX)
    SetThemeCursor(kThemeArrowCursor);

  // ..................................... lines drawn with system-supplied bit
patterns

  SetWTitle(gWindowRef,"\pClick mouse for more lines");
  QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
  while(!Button()) ;
  SetWTitle(gWindowRef,"\pDrawing lines with system-supplied bit patterns");
  FillRect(&portRect,&whitePattern);

  if(!gRunningOnX)
    SetThemeCursor(kThemeWatchCursor);

  for(a=1;a<39;a++)
  {
    b = doRandomNumber(0,portRect.bottom - portRect.top);
    c = doRandomNumber(0,portRect.bottom - portRect.top);
    theColor.red   = doRandomNumber(0,32767);
    theColor.green = doRandomNumber(0,32767);
    theColor.blue  = doRandomNumber(0,32767);
    RGBForeColor(&theColor);

    GetIndPattern(&systemPattern,sysPatListID,a);
    PenPat(&systemPattern);

    PenSize(1, a * 2);
```

```
    MoveTo(portRect.left,b);
    LineTo(portRect.right,c);

    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
    Delay(5,&finalTicks);
  }

  if(!gRunningOnX)
    SetThemeCursor(kThemeArrowCursor);

  // ..................................................... lines drawn with a pixel
pattern

  SetWTitle(gWindowRef,"\pClick mouse for more lines");
  QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
  while(!Button()) ;
  SetWTitle(gWindowRef,"\pDrawing lines with a pixel pattern");
  FillRect(&portRect,&whitePattern);

  if(!gRunningOnX)
    SetThemeCursor(kThemeWatchCursor);

  if(!(pixpatHdl = GetPixPat(rPixelPattern1)))
    ExitToShell();
  PenPixPat(pixpatHdl);


  for(a=1;a<60;a++)
  {
    b = doRandomNumber(0,portRect.right - portRect.left);
    c = doRandomNumber(0,portRect.right - portRect.left);

    PenSize(a * 2,1);

    MoveTo(b,portRect.top);
    LineTo(c,portRect.bottom);

    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
    Delay(5,&finalTicks);
  }

  DisposePixPat(pixpatHdl);

  SetClip(oldClipRgn);
  DisposeRgn(oldClipRgn);
  if(!gRunningOnX)
    SetThemeCursor(kThemeArrowCursor);

  // ..................................................... lines drawn with pattern mode
patXor

  SetWTitle(gWindowRef,"\pClick mouse for more lines");
  QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
  while(!Button()) ;
  SetWTitle(gWindowRef,"\pDrawing lines using pattern mode patXor");

  if(!gRunningOnX)
    SetThemeCursor(kThemeWatchCursor);

  RGBBackColor(&gRedColor);
  FillRect(&portRect,&whitePattern);

  PenSize(1,1);
  PenPat(GetQDGlobalsBlack(&blackPattern));
  PenMode(patXor);
```

```
  InsetRect(&portRect,10,10);

  for(a = portRect.left,b = portRect.right;a < portRect.right + 1;a++,b--)
  {
    MoveTo(a,portRect.top);
    LineTo(b,portRect.bottom);

    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
  }

  for(a = portRect.bottom,b = portRect.top;b < portRect.bottom + 1;a--,b++)
  {
    MoveTo(portRect.left,a);
    LineTo(portRect.right,b);

    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
  }

  if(!gRunningOnX)
    SetThemeCursor(kThemeArrowCursor);
}

// ************************************************************************
doFrameAndPaint

void  doFrameAndPaint(void)
{
  SInt16       a;
  Rect         portRect, theRect;
  Pattern      whitePattern;
  UInt32       finalTicks;
  Pattern      systemPattern;
  PixPatHandle pixpatHdl;
  PenNormal();
  PenSize(30,20);

  for(a=0;a<3;a++)
  {
    RGBBackColor(&gWhiteColor);
    GetWindowPortBounds(gWindowRef,&portRect);
    FillRect(&portRect,GetQDGlobalsWhite(&whitePattern));

    if(!gRunningOnX)
      SetThemeCursor(kThemeWatchCursor);

    // ..................................................................
preparation

    if(a == 0)
    {
      SetWTitle(gWindowRef,"\pFraming and painting with a color");
      RGBForeColor(&gRedColor);                              // set foreground color to red
    }
    else if(a == 1)
    {
      SetWTitle(gWindowRef,"\pFraming and painting with a bit pattern");

      RGBForeColor(&gBlueColor);                           // set foreground color to blue
      RGBBackColor(&gYellowColor);                        // set foreground color to yellow
      GetIndPattern(&systemPattern,sysPatListID,16);               // get bit pattern for
pen
      PenPat(&systemPattern);                                        // set pen bit
pattern
    }
    else if (a == 2)
    {
```

```
      SetWTitle(gWindowRef,"\pFraming and painting with a pixel pattern");

      if(!(pixpatHdl = GetPixPat(rPixelPattern1)))               // get pixel pattern for
pen
        ExitToShell();
      PenPixPat(pixpatHdl);                                          // set pen pixel
pattern
    }

    // ..................................................................... framing and
painting

    SetRect(&theRect,30,32,151,191);
    FrameRect(&theRect);                                                    //
FrameRect
    MoveTo(30,29);
    DrawString("\pFrameRect");
    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
    Delay(30,&finalTicks);

    OffsetRect(&theRect,140,0);
    FrameRoundRect(&theRect,30,50);                                       //
FrameRoundRect
    MoveTo(170,29);
    DrawString("\pFrameRoundRect");
    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
    Delay(30,&finalTicks);
    OffsetRect(&theRect,140,0);
    FrameOval(&theRect);                                                    //
FrameOval
    MoveTo(310,29);
    DrawString("\pFrameOval");
    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
    Delay(30,&finalTicks);

    OffsetRect(&theRect,140,0);
    FrameArc(&theRect,330,300);                                            //
FrameArc
    MoveTo(450,29);
    DrawString("\pFrameArc");
    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
    Delay(30,&finalTicks);

    OffsetRect(&theRect,-420,186);
    PaintRect(&theRect);                                                    //
PaintRect
    MoveTo(30,214);
    DrawString("\pPaintRect");
    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
    Delay(30,&finalTicks);

    OffsetRect(&theRect,140,0);
    PaintRoundRect(&theRect,30,50);                                       //
PaintRoundRect
    MoveTo(170,214);
    DrawString("\pPaintRoundRect");
    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
    Delay(30,&finalTicks);

    OffsetRect(&theRect,140,0);
    PaintOval(&theRect);                                                    //
PaintOval
    MoveTo(310,214);
    DrawString("\pPaintOval");
    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
    Delay(30,&finalTicks);
```

```
    OffsetRect(&theRect,140,0);
    PaintArc(&theRect,330,300);                                          //
PaintArc
    MoveTo(450,214);
    DrawString("\pPaintArc");
    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
    Delay(30,&finalTicks);

    if(!gRunningOnX)
      SetThemeCursor(kThemeArrowCursor);

    if(a < 2)
    {
      SetWTitle(gWindowRef,"\pClick mouse for more");
      QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
      while(!Button()) ;
    }
  }
  DisposePixPat(pixpatHdl);
}

// ************************************************************************
doFillEraseInvert

void  doFillEraseInvert(void)
{
  SInt16       a;
  Rect         portRect, theRect;
  Pattern      whitePattern, fillPat, backPat;
  PixPatHandle fillPixpatHdl, backPixpatHdl;
  UInt32       finalTicks;

  PenNormal();
  PenSize(30,20);

  for(a=0;a<4;a++)
  {
    if(a < 3)
    {
      RGBBackColor(&gWhiteColor);
      GetWindowPortBounds(gWindowRef,&portRect);
      FillRect(&portRect,GetQDGlobalsWhite(&whitePattern));
    }

    if(!gRunningOnX)
      SetThemeCursor(kThemeWatchCursor);

    // ..................................................................
preparation

    if(a == 0)
    {
      SetWTitle(gWindowRef,"\pFilling and erasing with colors");

      RGBForeColor(&gBlueColor);                          // set blue color for foreground
      RGBBackColor(&gRedColor);                            // set red color for background
      GetIndPattern(&fillPat,sysPatListID,1);      // get black bit pattern for fill
functions
      BackPat(&whitePattern);                          // set white bit pattern for
background
    }
    else if(a == 1)
    {
      SetWTitle(gWindowRef,"\pFilling and erasing with bit patterns");
```

```
      RGBForeColor(&gBlueColor);                          // set blue color for foreground
      RGBBackColor(&gYellowColor);                       // set yellow color for background
      GetIndPattern(&fillPat,sysPatListID,37);          // get bit pattern for fill
functions
      GetIndPattern(&backPat,sysPatListID,19);             // get bit pattern for
background
      BackPat(&backPat);                                   // set bit pattern for
background
    }
    else if (a == 2)
    {
      SetWTitle(gWindowRef,"\pFilling and erasing with pixel patterns");
      if(!(fillPixpatHdl = GetPixPat(rPixelPattern1)))     // get pixel patt - fill
functions
        ExitToShell();
      if(!(backPixpatHdl = GetPixPat(rPixelPattern2)))      // get pixel pattern -
background
        ExitToShell();
      BackPixPat(backPixpatHdl);                           // set pixel pattern -
background
    }
    else if(a == 3)
    {
      SetWTitle(gWindowRef,"\pInverting");

      BackPat(&whitePattern);
      SetRect(&theRect,30,15,570,29);
      EraseRect(&theRect);
      SetRect(&theRect,30,200,570,214);
      EraseRect(&theRect);
    }

    // ..................................................... filling, erasing, and
inverting

    SetRect(&theRect,30,32,151,191);
    MoveTo(30,29);
    if(a < 2)
    {
      FillRect(&theRect,&fillPat);                                          //
FillRect
      DrawString("\pFillRect");
    }
    else if(a == 2)
    {
      FillCRect(&theRect,fillPixpatHdl);                                    //
FillCRect
      DrawString("\pFillCRect");
    }
    else if(a == 3)
    {
      InvertRect(&theRect);                                                //
InvertRect
      DrawString("\pInvertRect");
    }
    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
    Delay(30,&finalTicks);

    OffsetRect(&theRect,140,0);
    MoveTo(170,29);
    if(a < 2)
    {
      FillRoundRect(&theRect,30,50,&fillPat);                             //
FillRoundRect
      DrawString("\pFillRoundRect");
    }
```

```
    else if(a == 2)
    {
      FillCRoundRect(&theRect,30,50,fillPixpatHdl);                          //
FillCRoundRect
      DrawString("\pFillCRoundRect");
    }
    else if(a == 3)
    {
      InvertRoundRect(&theRect,30,50);                                      //
InvertRoundRect
      DrawString("\pInvertRoundRect");
    }
    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
    Delay(30,&finalTicks);

    OffsetRect(&theRect,140,0);
    MoveTo(310,29);
    if(a < 2)
    {
      FillOval(&theRect,&fillPat);                                           //
FillOval
      DrawString("\pFillOval");
    }
    else if(a == 2)
    {
      FillCOval(&theRect,fillPixpatHdl);                                     //
FillCOval
      DrawString("\pFillCOval");
    }
    else if(a == 3)
    {
      InvertOval(&theRect);                                                 //
InvertOval
      DrawString("\pInvertOval");
    }
    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
    Delay(30,&finalTicks);

    OffsetRect(&theRect,140,0);
    MoveTo(450,29);
    if(a < 2)
    {
      FillArc(&theRect,330,300,&fillPat);                                     //
FillArc
      DrawString("\pFillArc");
    }
    else if(a == 2)
    {
      FillCArc(&theRect,330,300,fillPixpatHdl);                              //
FillCArc
      DrawString("\pFillCArc");
    }
    else if(a == 3)
    {
      InvertArc(&theRect,330,300);                                           //
InvertArc
      DrawString("\pInvertArc");
    }
    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
    Delay(30,&finalTicks);

    OffsetRect(&theRect,-420,186);
    MoveTo(30,214);
    if(a < 3)
    {
      EraseRect(&theRect);                                                   //
```

```
EraseRect
      DrawString("\pEraseRect");
    }
    else
    {
      InvertRect(&theRect);                                                    //
InvertRect
      DrawString("\pInvertRect");
    }
    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
    Delay(30,&finalTicks);

    OffsetRect(&theRect,140,0);
    MoveTo(170,214);
    if(a < 3)
    {
      EraseRoundRect(&theRect,30,50);                                          //
EraseRoundRect
      DrawString("\pEraseRoundRect");
    }
    else
    {
      InvertRoundRect(&theRect,30,50);                                        //
InvertRoundRect
      DrawString("\pInvertRoundRect");
    }
    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
    Delay(30,&finalTicks);

    OffsetRect(&theRect,140,0);
    MoveTo(310,214);
    if(a < 3)
    {
      EraseOval(&theRect);                                                     //
EraseOval
      DrawString("\pEraseOval");
    }
    else
    {
      InvertOval(&theRect);                                                    //
InvertOval
      DrawString("\pInvertOval");
    }
    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
    Delay(30,&finalTicks);

    OffsetRect(&theRect,140,0);
    MoveTo(450,214);
    if(a < 3)
    {
      EraseArc(&theRect,330,300);                                              //
EraseArc
      DrawString("\pEraseArc");
    }
    else
    {
      InvertArc(&theRect,330,300);                                             //
InvertArc
      DrawString("\pInvertArc");
    }
    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
    Delay(30,&finalTicks);

    if(!gRunningOnX)
      SetThemeCursor(kThemeArrowCursor);
    if(a < 3)
```

```
    {
      SetWTitle(gWindowRef,"\pClick mouse for more");
      QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
      while(!Button()) ;
    }
  }

  DisposePixPat(fillPixpatHdl);
  DisposePixPat(backPixpatHdl);
}

// ************************************************************************
doPolygonAndRegion

void  doPolygonAndRegion(void)
{
  Rect         portRect, theRect;
  Pattern      whitePattern, backPat;
  PixPatHandle fillPixpatHdl;
  PolyHandle   polygonHdl;
  RgnHandle    regionHdl;
  UInt32       finalTicks;

  SetWTitle(gWindowRef,"\pFraming, painting, filling, and erasing polygons and regions");

  RGBBackColor(&gWhiteColor);
  GetWindowPortBounds(gWindowRef,&portRect);
  FillRect(&portRect,GetQDGlobalsWhite(&whitePattern));

  if(!gRunningOnX)
    SetThemeCursor(kThemeWatchCursor);

  // ......................................................................
preparation

  GetIndPattern(&backPat,sysPatListID,17);                  // get bit pattern for
background
  BackPat(&backPat);                                        // set bit pattern for
background
  if(!(fillPixpatHdl = GetPixPat(rPixelPattern2)))     // get pixel pattern for fill
functions
    ExitToShell();
  RGBForeColor(&gRedColor);                                 // set red color for foreground
  RGBBackColor(&gYellowColor);                            // set yellow color for background
  PenNormal();

  polygonHdl = OpenPoly();                                                  // define
polygon
  MoveTo(30,32);
  LineTo(151,32);
  LineTo(96,103);
  LineTo(151,134);
  LineTo(151,191);
  LineTo(30,191);
  LineTo(66,75);
  ClosePoly();
  regionHdl = NewRgn();                                                      // define
region
  OpenRgn();
  SetRect(&theRect,30,218,151,279);
  FrameRect(&theRect);
  SetRect(&theRect,30,316,151,377);
  FrameRect(&theRect);
  SetRect(&theRect,39,248,142,341);
  FrameOval(&theRect);
  CloseRgn(regionHdl);
```

```
  // ............................................. framing, painting, filling, and
erasing

  FramePoly(polygonHdl);                                                    //
FramePoly
  MoveTo(30,29);
  DrawString("\pFramePoly (color)");
  QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
  Delay(30,&finalTicks);

  OffsetPoly(polygonHdl,140,0);
  PaintPoly(polygonHdl);                                                    //
PaintPoly
  MoveTo(170,29);
  DrawString("\pPaintPoly (color)");
  QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
  Delay(30,&finalTicks);

  OffsetPoly(polygonHdl,140,0);
  FillCPoly(polygonHdl,fillPixpatHdl);                                      //
FillCPoly
  MoveTo(310,29);
  DrawString("\pFillCPoly (pixel pattern)");
  QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
  Delay(30,&finalTicks);

  OffsetPoly(polygonHdl,140,0);
  ErasePoly(polygonHdl);                                                    //
ErasePoly
  MoveTo(450,29);
  DrawString("\pErasePoly (bit pattern)");
  QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
  Delay(30,&finalTicks);

  FrameRgn(regionHdl);                                                       //
FrameRgn
  MoveTo(30,214);
  DrawString("\pFrameRgn (color)");
  QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
  Delay(30,&finalTicks);

  OffsetRgn(regionHdl,140,0);
  PaintRgn(regionHdl);                                                       //
PaintRgn
  MoveTo(170,214);
  DrawString("\pPaintRgn (color)");
  QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
  Delay(30,&finalTicks);

  OffsetRgn(regionHdl,140,0);
  FillCRgn(regionHdl,fillPixpatHdl);                                         //
FillCRgn
  MoveTo(310,214);
  DrawString("\pFillCRgn (pixel pattern)");
  QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
  Delay(30,&finalTicks);

  OffsetRgn(regionHdl,140,0);
  EraseRgn(regionHdl);                                                       //
EraseRgn
  MoveTo(450,214);
  DrawString("\pEraseRgn (bit pattern)");
  QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
  Delay(30,&finalTicks);
```

```
  KillPoly(polygonHdl);
  DisposeRgn(regionHdl);
  DisposePixPat(fillPixpatHdl);
  BackPat(&whitePattern);

  if(!gRunningOnX)
    SetThemeCursor(kThemeArrowCursor);
}

// ********************************************************************************
doText

void  doText(void)
{
  Rect    portRect, theRect;
  Pattern whitePattern;
  SInt16  windowCenter, a, fontNum, stringWidth;
  Str255  textString;
  UInt32  finalTicks;

  RGBBackColor(&gWhiteColor);
  GetWindowPortBounds(gWindowRef,&portRect);
  FillRect(&portRect,GetQDGlobalsWhite(&whitePattern));

  SetWTitle(gWindowRef,"\pDrawing text with default source mode (srcOr)");

  windowCenter = (portRect.right - portRect.left) / 2;
  SetRect(&theRect,windowCenter,portRect.top,portRect.right,portRect.bottom);
  RGBBackColor(&gBlueColor);
  FillRect(&theRect,&whitePattern);

  if(!gRunningOnX)
    SetThemeCursor(kThemeWatchCursor);

  for(a=1;a<9;a++)
  {
    // ........................... set various text fonts, text styles, and foreground
colors

    if(a == 1)
    {
      GetFNum("\pGeneva",&fontNum);
      TextFont(fontNum);
      TextFace(normal);
      RGBForeColor(&gRedColor);
    }
    else if(a == 2)
      TextFace(bold);
    else if(a == 3)
    {
      GetFNum("\pTimes",&fontNum);
      TextFont(fontNum);
      TextFace(italic);
      RGBForeColor(&gYellowColor);
    }
    else if(a == 4)
      TextFace(underline);
    else if(a == 5)
    {
      GetFNum("\pHelvetica",&fontNum);
      TextFont(fontNum);
      TextFace(normal);
      RGBForeColor(&gGreenColor);
    }
    else if(a == 6)
      TextFace(bold + italic);
```

```
    else if(a == 7)
    {
      GetFNum("\pCharcoal",&fontNum);
      TextFont(fontNum);
      TextFace(condense);
      RGBForeColor(&gBlackColor);
    }
    else if(a == 8)
    {
      TextFace(extend);
    }

    // ..................................................................... set text
size

    if(a < 7)
      TextSize(a * 2 + 15);
    else
      TextSize(12);

    // ........... get a string and draw it in the set font, style, size, and foreground
color

    GetIndString(textString,rFontsStringList,a);
    stringWidth = StringWidth(textString);
    MoveTo(windowCenter - (stringWidth / 2),a * 46 - 10);
    DrawString(textString);

    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
    Delay(30,&finalTicks);
  }
  // ..................................................... reset to Geneva 10pt
normal

  GetFNum("\pGeneva",&fontNum);
  TextFont(fontNum);
  TextSize(10);
  TextFace(normal);

  // ............ erase a rectangle, get a string, and use TETextBox to draw it left
justified

  SetRect(&theRect,portRect.left + 5,portRect.bottom - 55,portRect.left + 118,
          portRect.bottom - 5);
  EraseRect(&theRect);
  InsetRect(&theRect,5,5);
  GetIndString(textString,rFontsStringList,9);
  RGBForeColor(&gWhiteColor);
  TETextBox(&textString[1],textString[0],&theRect,teFlushLeft);

  if(!gRunningOnX)
    SetThemeCursor(kThemeArrowCursor);
}

// ****************************************************************************
doScrolling

void  doScrolling(void)
{
  Rect         portRect, theRect;
  Pattern      whitePattern;
  PixPatHandle pixpat1Hdl, pixpat2Hdl;
  RgnHandle    oldClipHdl, regionAHdl, regionBHdl, regionCHdl, scrollRegionHdl;
  SInt16       a;
  UInt32       finalTicks;
```

```
SetWTitle(gWindowRef,"\pScrolling pixels");

RGBBackColor(&gWhiteColor);
GetWindowPortBounds(gWindowRef,&portRect);
FillRect(&portRect,GetQDGlobalsWhite(&whitePattern));

if(!gRunningOnX)
  SetThemeCursor(kThemeWatchCursor);

if(!(pixpat1Hdl = GetPixPat(rPixelPattern1)))
  ExitToShell();
PenPixPat(pixpat1Hdl);
PenSize(50,0);
SetRect(&theRect,30,30,286,371);
FrameRect(&theRect);
SetRect(&theRect,315,30,571,371);
FillCRect(&theRect,pixpat1Hdl);

if(!(pixpat2Hdl = GetPixPat(rPixelPattern2)))
  ExitToShell();
BackPixPat(pixpat2Hdl);
regionAHdl = NewRgn();
regionBHdl = NewRgn();
regionCHdl = NewRgn();
SetRect(&theRect,80,30,236,371);
RectRgn(regionAHdl,&theRect);
SetRect(&theRect,315,30,571,371);
RectRgn(regionBHdl,&theRect);
UnionRgn(regionAHdl,regionBHdl,regionCHdl);

oldClipHdl = NewRgn();
GetClip(oldClipHdl);
SetClip(regionCHdl);

SetRect(&theRect,80,30,571,371);

scrollRegionHdl = NewRgn();

for(a=0;a<371;a++)
{
  ScrollRect(&theRect,0,1,scrollRegionHdl);
  QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
  theRect.top ++;
  Delay(1,&finalTicks);
}

SetRect(&theRect,80,30,571,371);
BackPixPat(pixpat1Hdl);

for(a=0;a<371;a++)
{
  ScrollRect(&theRect,0,-1,scrollRegionHdl);
  QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
  theRect.bottom --;
  Delay(1,&finalTicks);
}

SetClip(oldClipHdl);

DisposePixPat(pixpat1Hdl);
DisposePixPat(pixpat2Hdl);
DisposeRgn(oldClipHdl);
DisposeRgn(regionAHdl);
DisposeRgn(regionBHdl);
DisposeRgn(regionCHdl);
DisposeRgn(scrollRegionHdl);
```

```
  if(!gRunningOnX)
    SetThemeCursor(kThemeArrowCursor);
}
// *****************************************************************************
doBooleanSourceModes

void  doBooleanSourceModes(void)
{
  Rect         portRect, theRect;
  Pattern      whitePattern;
  Handle       destIconHdl, sourceIconHdl;
  SInt16       a, b;
  UInt32       finalTicks;
  BitMap       sourceIconMap;
  Str255       sourceString;
  PixMapHandle destinationPixMapHdl;

  SetWTitle(gWindowRef,"\pBoolean source modes");

  RGBForeColor(&gBlackColor);
  RGBBackColor(&gGreenColor);
  GetWindowPortBounds(gWindowRef,&portRect);
  FillRect(&portRect,GetQDGlobalsWhite(&whitePattern));
  SetRect(&theRect,portRect.left,portRect.top,portRect.right,
          (portRect.bottom - portRect.top) / 2);
  RGBBackColor(&gWhiteColor);
  FillRect(&theRect,&whitePattern);

  if(!gRunningOnX)
    SetThemeCursor(kThemeWatchCursor);

  destIconHdl = GetIcon(rDestinationIcon);
  sourceIconHdl = GetIcon(rSourceIcon);

  for(a=0;a<2;a++)
  {
    if(a == 1)
    {
      RGBForeColor(&gYellowColor);
      RGBBackColor(&gRedColor);
    }

    SetRect(&theRect,235,a * 191 + 30,299,a * 191 + 94);
    PlotIcon(&theRect,destIconHdl);
    MoveTo(235,a * 191 + 27);
    DrawString("\pDestination");

    SetRect(&theRect,304,a * 191 + 30,368,a * 191 + 94);
    PlotIcon(&theRect,sourceIconHdl);
    MoveTo(304,a * 191 + 27);
    DrawString("\pSource");
  }

  RGBForeColor(&gBlackColor);
  RGBBackColor(&gWhiteColor);
  for(a=0;a<2;a++)
  {
    if(a == 1)
    {
      RGBForeColor(&gYellowColor);
      RGBBackColor(&gRedColor);
    }

    for(b=0;b<8;b++)
    {
```

```
      SetRect(&theRect,b * 69 + 28,a * 191 + 121,b * 69 + 92,a * 191 + 185);
      PlotIcon(&theRect,destIconHdl);
    }
  }

  QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);

  RGBForeColor(&gBlackColor);
  RGBBackColor(&gWhiteColor);

  HLock(sourceIconHdl);
  sourceIconMap.baseAddr = *sourceIconHdl;
  sourceIconMap.rowBytes = 4;
  SetRect(&sourceIconMap.bounds,0,0,32,32);

  destinationPixMapHdl = GetPortPixMap(GetQDGlobalsThePort());

  for(a=0;a<2;a++)
  {
    if(a == 1)
    {
      RGBForeColor(&gYellowColor);
      RGBBackColor(&gRedColor);
    }

    for(b=0;b<8;b++)
    {
      Delay(30,&finalTicks);
      SetRect(&theRect,b * 69 + 28,a * 191 + 121,b * 69 + 92,a * 191 + 185);

      CopyBits(&sourceIconMap,
               (BitMap *) *destinationPixMapHdl,
               &sourceIconMap.bounds,
               &theRect,
               b,NULL);
      GetIndString(sourceString,rBooleanStringList,b + 1);
      MoveTo(b * 69 + 28,a * 191 + 118);
      DrawString(sourceString);

      QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
    }
  }

  HUnlock(sourceIconHdl);
  if(!gRunningOnX)
    SetThemeCursor(kThemeArrowCursor);
}

// ****************************************************************
doArithmeticSourceModes

void  doArithmeticSourceModes(void)
{
  Rect         portRect, sourceRect, destRect;
  Pattern      whitePattern;
  PicHandle    sourceHdl, destinationHdl;
  SInt16       a, b, arithmeticMode = 32;
  PixMapHandle currentPixMapHdl;
  Str255       modeString;
  UInt32       finalTicks;

  SetWTitle(gWindowRef,"\pCopyBits with arithmetic source modes");

  RGBForeColor(&gBlackColor);
  RGBBackColor(&gWhiteColor);
  GetWindowPortBounds(gWindowRef,&portRect);
```

```
  FillRect(&portRect,GetQDGlobalsWhite(&whitePattern));

  if(!gRunningOnX)
    SetThemeCursor(kThemeWatchCursor);

  if(!(sourceHdl = GetPicture(rPicture)))
    ExitToShell();
  SetRect(&sourceRect,44,21,201,133);
  HNoPurge((Handle) sourceHdl);
  DrawPicture(sourceHdl,&sourceRect);
  HPurge((Handle) sourceHdl);
  MoveTo(44,19);
  DrawString("\pSOURCE IMAGE");

  if(!(destinationHdl = GetPicture(rPicture + 1)))
    ExitToShell();
  HNoPurge((Handle) destinationHdl);
  for(a=44;a<403;a+=179)
  {
    for(b=21;b<274;b+=126)
    {
      if(a == 44 && b == 21)
       continue;
      SetRect(&destRect,a,b,a+157,b+112);
      DrawPicture(destinationHdl,&destRect);
    }
  }
  HPurge((Handle) destinationHdl);

  QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);

  currentPixMapHdl = GetPortPixMap(GetWindowPort(gWindowRef));
  for(a=44;a<403;a+=179)
  {
    for(b=21;b<274;b+=126)
    {
      if(a == 44 && b == 21)
       continue;

      Delay(60,&finalTicks);

      GetIndString(modeString,rArithmeticStringList,arithmeticMode - 31);
      MoveTo(a,b - 2);
      DrawString(modeString);

      SetRect(&destRect,a,b,a+157,b+112);

      CopyBits((BitMap *) *currentPixMapHdl,
               (BitMap *) *currentPixMapHdl,
               &sourceRect,&destRect,
               arithmeticMode + ditherCopy,NULL);

      arithmeticMode ++;

      QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
    }

  }

  ReleaseResource((Handle) sourceHdl);
  ReleaseResource((Handle) destinationHdl);

  if(!gRunningOnX)
    SetThemeCursor(kThemeArrowCursor);
}
```

```
// ****************************************************************************
doHighlighting

void  doHighlighting(void)
{
  Rect     portRect, theRect;
  Pattern  whitePattern;
  RGBColor oldHighlightColor;
  SInt16   a;
  UInt8    hiliteVal;
  UInt32   finalTicks;

  SetWTitle(gWindowRef,"\pHighlighting");

  RGBForeColor(&gBlackColor);
  RGBBackColor(&gWhiteColor);
  GetWindowPortBounds(gWindowRef,&portRect);
  FillRect(&portRect,GetQDGlobalsWhite(&whitePattern));

  LMGetHiliteRGB(&oldHighlightColor);
  for(a=0;a<3;a++)
  {
    MoveTo(50,a * 100 + 60);
    DrawString("\pClearing the highlight bit and calling InvertRect.");
    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
    Delay(60,&finalTicks);
    SetRect(&theRect,44,a * 100 + 44,557,a * 100 + 104);

    hiliteVal = LMGetHiliteMode();
    BitClr(&hiliteVal,pHiliteBit);
    LMSetHiliteMode(hiliteVal);

    if(a == 1)
      HiliteColor(&gYellowColor);
    else if(a == 2)
      HiliteColor(&gGreenColor);

    InvertRect(&theRect);
    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);

    MoveTo(50,a * 100 + 75);
    Delay(60,&finalTicks);
    DrawString("\pClick mouse to unhighlight. ");
    DrawString("\p(Note:  The call to InvertRect reset the highlight bit ...");
    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);

    while(!Button()) ;

    MoveTo(45,a * 100 + 90);
    DrawString("\p... so we clear the highlight bit again before calling InvertRect.)");
    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
    Delay(60,&finalTicks);

    LMSetHiliteMode(hiliteVal);

    InvertRect(&theRect);
    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
  }

  HiliteColor(&oldHighlightColor);

  Delay(60,&finalTicks);
  MoveTo(50,350);
  DrawString("\pOriginal highlight color has been reset.");

  QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
```

```
}

// ************************************************************************
doDrawWithMouse

void  doDrawWithMouse(void)
{
  Rect         portRect, drawRect;
  Pattern      whitePattern, blackPattern;
  PixPatHandle pixpatHdl;
  Point        initialMouse, previousMouse, currentMouse;
  UInt16       randomNumber;
  RGBColor     theColor;

  RGBBackColor(&gWhiteColor);
  GetWindowPortBounds(gWindowRef,&portRect);
  FillRect(&portRect,GetQDGlobalsWhite(&whitePattern));

  if(!(pixpatHdl = GetPixPat(rPixelPattern3)))
    ExitToShell();
  PenPixPat(pixpatHdl);
  PenSize(1,1);
  PenMode(patXor);

  GetMouse(&initialMouse);
  drawRect.left = drawRect.right  = initialMouse.h;
  drawRect.top  = drawRect.bottom = initialMouse.v;

  GetMouse(&previousMouse);

  while(StillDown())
  {
    GetMouse(&currentMouse);

    if(currentMouse.v != previousMouse.v || currentMouse.h != previousMouse.h)
    {
      FrameRect(&drawRect);

      if(currentMouse.h >= initialMouse.h)
        drawRect.right = currentMouse.h;
      if(currentMouse.v >= initialMouse.v)
        drawRect.bottom = currentMouse.v;
      if(currentMouse.h <= initialMouse.h)
        drawRect.left = currentMouse.h;
      if(currentMouse.v <= initialMouse.v)
        drawRect.top = currentMouse.v;

      FrameRect(&drawRect);
    }

    previousMouse.v = currentMouse.v;
    previousMouse.h = currentMouse.h;
  }

  FrameRect(&drawRect);

  theColor.red   = doRandomNumber(0,65535);
  theColor.green = doRandomNumber(0,65535);
  theColor.blue  = doRandomNumber(0,65535);
  RGBForeColor(&theColor);

  PenMode(patCopy);
  PenPat(GetQDGlobalsBlack(&blackPattern));
  BackPixPat(pixpatHdl);
  randomNumber = doRandomNumber(0,3);
```

```
  if(randomNumber == 0)
    PaintRect(&drawRect);
  else if(randomNumber == 1)
    EraseRoundRect(&drawRect,50,50);
  else if(randomNumber == 2)
    PaintOval(&drawRect);
  else if(randomNumber == 3)
    PaintArc(&drawRect,0,doRandomNumber(0,360));

  BackPat(&whitePattern);
}

// ****************************************************************************
doDrawingState

void  doDrawingState(void)
{
  Rect              portRect, theRect;
  Pattern           whitePattern;
  ThemeDrawingState themeDrawingState;
  SInt16            a;
  UInt32            finalTicks;

  RGBBackColor(&gBlueColor);
  GetWindowPortBounds(gWindowRef,&portRect);
  FillRect(&portRect,GetQDGlobalsWhite(&whitePattern));
  SetWTitle(gWindowRef,"\pSaving and restoring the graphics port drawing state");

  if(!gRunningOnX)
    SetThemeCursor(kThemeWatchCursor);

  NormalizeThemeDrawingState();

  doDrawingStateProof(0);
  Delay(120,&finalTicks);

  GetThemeDrawingState(&themeDrawingState);

  theRect = portRect;
  theRect.right -= 300;

  SetThemeBackground(kThemeBrushListViewBackground,gPixelDepth,gIsColorDevice);
  EraseRect(&theRect);

  theRect.left += 150;

  SetThemeBackground(kThemeBrushListViewSortColumnBackground,gPixelDepth,gIsColorDevice);
  EraseRect(&theRect);

  SetThemePen(kThemeBrushListViewSeparator,gPixelDepth,gIsColorDevice);

  theRect.left -= 150;
  for(a=theRect.top;a<=theRect.bottom;a+=18)
  {
    MoveTo(theRect.left,a);
    LineTo(theRect.right - 1,a);
  }

  Delay(120,&finalTicks);
  doDrawingStateProof(1);
  Delay(120,&finalTicks);

  SetThemeDrawingState(themeDrawingState,true);

  doDrawingStateProof(2);
```

```
  if(!gRunningOnX)
    SetThemeCursor(kThemeArrowCursor);
}

// ************************************************************************
doDrawingStateProof

void  doDrawingStateProof(SInt16 phase)
{
  Rect theRect;

  MoveTo(324,phase * 117 + 41);
  if(phase == 0)
    DrawString("\pBefore calls to SetThemePen and SetThemeBackground");
  else if(phase == 1)
    DrawString("\pAfter calls to SetThemePen and SetThemeBackground");
  else if(phase == 2)
    DrawString("\pAfter restoration of graphics port drawing state");

  MoveTo(324,phase * 117 + 54);
  DrawString("\pPen pattern/color");
  MoveTo(462,phase * 117 + 54);
  DrawString("\pBackgrd pattern/color");

  SetRect(&theRect,324,phase * 117 + 58,438,phase * 117 + 132);
  PaintRect(&theRect);
  SetRect(&theRect,462,phase * 117 + 58,576,phase * 117 + 132);
  EraseRect(&theRect);

  QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
}

// ************************************************************************
doGetDepthAndDevice

void doGetDepthAndDevice(void)
{
  GDHandle deviceHdl;

  deviceHdl = GetMainDevice();
  gPixelDepth = (*(*deviceHdl)->gdPMap)->pixelSize;
  if(((1 << gdDevType) & (*deviceHdl)->gdFlags) != 0)
    gIsColorDevice = true;
}
// ************************************************************************
doRandomNumber

UInt16  doRandomNumber(UInt16 minimum,UInt16 maximum)
{
  UInt16 randomNumber;
  SInt32 range, t;

  randomNumber = Random();
  range = maximum - minimum + 1;
  t = (randomNumber * range) / 65536;
  return (t + minimum);
}

//
****************************************************************************************
```

Demonstration Program QuickDraw Comments

When this program is run, the user should choose items from the Demonstration menu and click the mouse button when instructed to do so by the advisory text in the window's title bar.

defines

In addition to the usual constants relating to menus and the window, constants are established for pixel pattern, icon, string list, and picture resource IDs.

Global Variables

The fields of the RGBColor global variables are assigned values representing the colors described by the variable names.

main

Random numbers are used by various functions in the demonstration. The call to `SetQDGlobalsRandomSeed` seeds the random number generator. `randSeed` is a QuickDraw global variable which holds the seed value for the random number generator. Unless `randSeed` is modified, the same sequence of numbers will be generated each time the program is run. The parameter to the `GetDateTime` call receives the number of seconds since midnight, January 1, 1904, a value that is bound to be different each time the program is run.

Note that error handling in `main()`, as in other areas of the program, is somewhat rudimentary in that the program simply terminates.

doEvents

Within the `mouseDown case`, at the `inContent` case, if the `mouseDown` is within the content region of the window when it is the front window and `gDrawWithMouseActivated` is true, the function `doDrawWithMouse` is called.

doDemonstrationMenu

`doDemonstrationMenu` switches according to the user's choices in the Demonstration menu. In all but the `iDrawWithMouse` case, the only action taken is to call the relevant function.

Note that the global variable `gDrawWithMouseActivated` is set to false at function entry, and is set to true within the `iDrawWithMouse` case (which executes if the user chooses the Draw with Mouse item). Also note that the window's background is filled with the white color, using the white pattern, within this case.

doLines

`doLines` demonstrates line drawing using colors, bit patterns, pixel patterns, and with the Boolean pattern mode `patXor`. `doLines` also demonstrates modifying the graphics port's clipping region so as to clip drawing to that modified region.

The first line sets the graphics pen's size, pattern, and pattern mode to the defaults. The next three lines fill the window's content area with blue.

The next block sets the window's clipping region to a rectangle 10 pixels inside the port rectangle. The first two lines define such a rectangle. The next two lines save the current clipping region for later restoration. The call to `ClipRect` establishes the new clipping region by setting it in the graphics port object.

Lines Drawn with Foreground Color and Black Pen Pattern

After the window title is set, `FillRect` is called with the white pattern with the background color is set to white. This fill is clipped to the current clipping region, which is a rectangle 10 pixels inside the port rectangle.

Within the for loop, random numbers between 0 and the width of the port rectangle are assigned to two variables which will be used to specify the starting and finishing horizontal coordinates for each of 60 drawn lines. The fields of an RGBColor variable are also assigned random values, this time between 0 and 65534 (one less than the maximum possible value for a UInt16). The call to RGBColor assigns this random color as the requested foreground color. The pen width is increased by two pixels. Finally, the call to `MoveTo` moves the pen to the random horizontal location at the top of the port rectangle, and the call to `LineTo` draws a line to the random horizontal location at the bottom of the port rectangle. The line drawing is clipped to the current clipping region.

Lines Drawn with System-Supplied Bit Patterns

This line drawing operation is similar to the previous one except that a system-supplied bit pattern is assigned to the graphics pen and the lines are drawn from left to right rather than top to bottom. The bit patterns are loaded by the call to `GetIndPattern` and are drawn from the 38 patterns in the `'PAT#'` resource in the System file with resource ID `sysPatListID` (0). The call to `PenPat` assigns the specified bit pattern to the graphics pen. In this operation, the height of the pen, rather than the width, is increased by two each

time around the for loop.

Lines Drawn with a Pixel Pattern

In this line drawing operation, before the for loop is entered, `GetPixPat` is called to allocate a `PixPat` structure and initialize it with information from the specified `'ppat'` resource. The call to `PenPixPat` then assigns this pixel pattern to the graphics pen.

After the last line is drawn, `DisposePixPat` is called to free the memory allocated by the `GetPixPat` call.

At this point, the clipping region saved at the start of the function is restored, and all of the memory allocated by the `NewRgn` call is freed.

Lines Drawn with Pattern Mode patXor

This block demonstrates a well-known but nonetheless exotic capability of the humble line when it operates in the pattern mode `patXor`.

The content area is filled with red, following which the pen size and pen pattern are set to the defaults. The call to `PenMode` sets the pen mode to `patXor`. The next four lines assign values to four variables that will be used to ensure that the starting and ending locations of each drawn line will be ten pixels inside the port rectangle. The for loops, proceeding clockwise, draw lines from points 10 pixels inside the periphery of the port rectangle through the center of the rectangle to points on the opposite side of the rectangle. The effect of `patXor` on any destination pixel is to invert it. For example, assuming a white background and black pen color, any white pixel in the path of the drawn lines will be turned black and any black pixel will be turned white. This produces a pattern known as a moire (watered silk) pattern.

doFrameAndPaint

`doFrameAndPaint` demonstrates the use of QuickDraw's framing and painting functions with the exception of those relating to polygons and regions.

At the first two lines, the pen pattern and mode are set to the defaults and the pen size is set to 30 pixels wide and 20 pixels high.

The for loop is traversed three times, once for framing and painting with a color, once for framing and painting with a bit pattern, and once for framing and painting with a pixel pattern. The first action is to fill the port rectangle with the color white using the white pattern.

Preparation

The first time around the loop, `RGBForeColor` is called to set the requested foreground color to red.

The second time around the loop, `RGBForeColor` and `RGBBackColor` are called to set the requested foreground and background colors to, respectively, blue and yellow, `GetIndPattern` loads one of the system-supplied bit patterns, and `PenPat` makes that pattern the pen's current bit pattern.

The third time around the loop, a call to `GetPixPat` loads a `'ppat'` resource, creating a new `PixPat` structure, and a call to `PenPixPat` assigns that pixel pattern to the pen.

Framing and Painting

In this section, `SetRect` is used to assign the coordinates of a rectangle to the fields of a `Rect` structure, and `OffsetRect` is used to move the rectangle horizontally and vertically between the calls to the various framing and painting functions.

Before `doFrameAndPaint` exits, `DisposePixPat` is called to free the memory allocated by the `GetPixPat` call.

doFillEraseInvert

`doFillEraseInvert` demonstrates the use of QuickDraw's filling, erasing, and inverting functions with the exception of those relating to polygons and regions.

At the first two lines, the pen pattern and mode are set to the defaults and the pen size is set to 30 pixels wide and 20 pixels high.

The for loop is traversed four times, once for filling and erasing with colors, once for filling and erasing with bit patterns, once for filling and erasing with a pixel patterns, and once for inverting. The first action, on the first three passes only, is to fill the port rectangle with the color white using the white pattern.

Preparation

The first time around the loop, `RGBForeColor` and `RGBBackColor` are called to set the requested foreground and background colors to, respectively, blue and red. In addition, the calls to `GetIndPattern` and `BackPat` set the background pattern to black.

The second time around the loop, `RGBForeColor` and `RGBBackColor` are called to set the requested foreground and background colors to, respectively, blue and yellow. In addition, `GetIndPattern` is called twice, once to assign a bit pattern to a Pattern

variable which will be passed as the second parameter in calls to `FillRect`, `FillOval`, and so on, and once, in conjunction with `BackPat`, to assign a bit pattern to the graphics port's `bkPixPat` field.

The third time around the loop, `GetPixPat` is called twice, once to assign a pixel pattern to a the variable which will be passed as the second parameter in calls to `FillCRect`, `FillCOval`, and so on, and once, in conjunction with `BackPixPat`, to assign a pixel pattern to the graphics port's `bkPixPat` field.

The fourth time around the loop, and preparatory to calls to the erasing functions, the call to `BackPat` sets the background pattern to white. (The calls to `SetRect` and `EraseRect` simply erase the existing text in the window.)

Filling, Erasing, and Inverting

In this section, `SetRect` is used to assign the coordinates of a rectangle to the fields of a Rect structure, and `OffsetRect` is used to move the rectangle horizontally and vertically between the calls to the various filling, erasing, and inverting functions.

Before `doFillEraseInvert` exits, `DisposePixPat` is called twice to free the memory allocated by the two `GetPixPat` calls.

doPolygonAndRegion

`doPolygonAndRegion` demonstrates defining a polygon and a region and the use of some of QuickDraw's polygon and region framing, painting, filling, and erasing functions.

Preparation

The calls to `GetIndPattern` and `BackPat` set the background pattern to one on the system- supplied bit patterns. The call to `GetPixPat` gets the pixel pattern to be used by the filling functions. The calls to `RGBForeColor` and `RGBBackColor` set the requested foreground and background colors. `PenNormal` sets the pen's size, pattern mode, and pattern to the defaults.

The `OpenPoly` call initiates the recording of the polygon definition, the `MoveTo` and `LineTo` calls define the polygon, and `ClosePoly` stops the recording. Note that, in this demonstration, the last vertex is not joined to the first vertex.

The `NewRgn` call allocates memory for a new region and a region pointer, initializes the contents of the region and make it an empty rectangle. `OpenRgn` initiates the recording of a region shape. The next seven lines create a region definition comprising two rectangles and an overlapping oval. `CloseRgn` terminates the recording.

Framing, Painting, Filling, and Erasing

In this section, `OffsetPoly` and `OffsetRgn` are used to move the polygon and region horizontally between the calls to the framing, filling, and erasing functions. `OffsetPoly` modifies the polygon's definition. `OffsetRgn` adjusts the coordinates of the region.

Before `doPolygonAndRegion` exits, `KillPoly` is called to free all the memory allocated by `OpenPoly`, `DisposeRgn` is called to free all the memory allocated by `NewRgn`, `DisposePixPat` is called to free all the memory allocated by `GetPixPat`, and the background pattern is set to white.

doText

`doText` draws text in various fonts, sizes and styles. In addition, the last block demonstrates drawing justified text within a specified rectangle using the `TextEdit` function `TETextBox`.

Prior to the for loop, the variable `windowCenter` is assigned a value which represents a location midway across the port rectangle, and the right half of the content area is filled with blue.

Within the first section of the for loop, the text font is changed using `GetFNum` and `TextFont`, the text style is changed using `TextFace`, and the foreground color is changed. At the last two sections within the loop, the text size is changed using `TextSize`, a string is retrieved from a `'STR#'` resource, the width of the string in pixels is determined, and the string is drawn centered laterally in the window.

After the loop exits, the text font, size and style are returned to Geneva 10pt plain.

At the final block, a small rectangle is defined at the bottom left of the content area. Because the current background color is blue, the call to EraseRect erases the rectangle in that color. The rectangle is then inset by five pixels all round. A string is then loaded from a `'STR#'` resource and the foreground color is set to white. Finally, `TETextBox` is called to draw the text within the specified rectangle with left justification. (Other available justification constants are `teFlushRight` and `teCenter`.)

doScrolling

`doScrolling` demonstrates scrolling pixels within a specified rectangle, with the operation clipped to a region comprising two unconnected rectangular areas.

The first call to `GetPixPat` loads a `'ppat'` resource. The call to `PenPixPat` assigns that pixel pattern to the pen, which is then made 50 pixels wide and zero pixels high. A framed rectangle is then drawn in the left half of the window. (Note that, because the pen

height is set to zero, the two sides of the rectangle will be drawn but not the top and bottom.) A filled rectangle is then drawn in the right side of the window using the same pixel pattern.

In the next block, another `'ppat'` resource is retrieved. The call to `BackPixPat` makes this pixel pattern the background pixel pattern.

The next block creates a region comprising two separate rectangles, the first one coincident with the "inside" of the framed rectangle and the second one coincident with the whole of the filled rectangle). The current clipping region is then saved and the newly created region is established as the current clipping region.

The following call to `SetRect` defines a rectangle for the first parameter of the `ScrollRect` function. Laterally, this extends from the left inside of the framed rectangle to the right hand side of the filled rectangle. The call to `NewRgn` then creates the empty region required by the `ScrollRect` calls.

In the first for loop, the pixels within the clipping region within the specified rectangle are scrolled down, the top of the rectangle being incremented down between calls to `ScrollRect`. `ScrollRect` fills the "vacated" areas with the background pattern.

Between the for loops, the rectangle used by `ScrollRect` is redefined and the background pixel pattern is changed to the pixel pattern used to draw the original rectangles. The scrolling operation is then repeated, this time in an upward direction.

Before `doScrolling` exits, the saved clipping region is restored and all the memory allocated by the `GetPixPat` and `NewRgn` calls is freed.

doBooleanSourceModes

`doBooleanSourceModes` demonstrates the effects of the Boolean source modes in both black-and-white and color.

The first block fills the content area with green and then fills the top half of the content area with white. This block leaves the foreground color black and the background color white.

The next block loads two 32-bit by 32-bit `'ICON'` resources. One icon contains the image of a cross and the other contains the image of a square.

The first for loop calls `PlotIcon` four times, twice to draw the icons in the white area at the top of the window, and twice to draw them in the green area at the bottom of the window. The rectangle passed in the first parameter of the `PlotIcon` calls expands the icon to 64 pixels by 64 pixels. The calls to `RGBForeColor` and `RGBBackColor` cause

the icons in the green area to be drawn using a foreground color of yellow and a background color of red.

The foreground and background colors are reset to black and white before the second for loop is entered.

The second for loop draws the cross icon eight times across the bottom of the white half of the window. The foreground and background colors are then changed to yellow and red before this process is repeated across the bottom of the green area of the window.

The foreground and background colors are again reset to black and white.

As a preamble to what is to come, note that there is no special data type for an icon. It is simply 128 bytes of bit data arranged as 32 rows of 4 bytes per row. All that is available is a handle to that 128 bytes of data. The intention is to cause the 128 bytes of data which constitutes the square icon to be regarded as bitmap data pointed to by the `baseAddr` field of a `BitMap` structure. That way, the `CopyBits` routine can be used to copy the bitmap into the graphics port.

Because `CopyBits` is one of those functions which can move memory around, the first action is to lock the icon data in the heap. The address of the square icon image data is then assigned to the `baseAddr` field of a `BitMap` structure, the `rowBytes` field is assigned the value 4, and the bounds field is assigned a rectangle defining the normal icon size.

The final for loop calls `CopyBits` to copy the bit image into the graphics port sixteen times, overdrawing the previously drawn cross icons. The call to `SetRect` within the inner for loop defines the expanded destination rectangle that governs the size at which the image will be drawn. This rectangle is passed in the `destRect` parameter of the `CopyBits` call. Note that, in the `CopyBits` call, the value passed in the `tMode` (transfer mode) parameter is incremented each time through the loop so that the square image overdraws the cross image once in each of the eight available Boolean source modes. The three lines following the `CopyBits` call retrieve the appropriate string containing the relevant source mode from the specified `'STR#'` resource and draw this string above each copied image.

The last line unlocks the icon image data.

doArithmeticSourceModes

`doArithmeticSourceModes` demonstrates the effects of the arithmetic source modes.

Since `CopyBits` will be called, the foreground and background colors are set to black and white respectively. The call to `FillRect` clears the window to white.

The first call to `GetPicture` loads a `'PICT'` resource into a Picture structure. (Because the `'PICT'` resource is purgeable, it is made non-purgeable as soon as it is retrieved, used immediately, and immediately made purgeable again.) The call to `DrawPicture` draws the picture in the top left of the window, where it is labeled as the source image.

The second call to `GetPicture` loads another `'PICT'` resource that will be used as the destination image. The first for loop draws this picture in the window at eight separate locations, these locations being determined by the rectangle passed in the first parameter of the `DrawPicture` calls.

The last for loop is traversed once for each of the eight arithmetic source modes. `CopyBits` is called eight times to overdraw the destination images with the source image. Note that the value in the `tMode` (transfer mode) parameter of the `CopyBits` call is incremented each time around the loop. Note also that each time around the loop a new string is retrieved from a `'STR#'` resource and drawn above the destination image.

Before `doArithmeticSourceModes` exits, `ReleaseResource` is called twice to free the memory allocated by the `GetPicture` calls.

doHighlighting

`doHighlighting` demonstrates highlighting, first with the color set by the user in the Appearance pane of the Appearance control panel (Mac OS 8/9) or in System Preferences (Mac OS X), and then with two colors set by the program.

The highlight color set by the user is saved via a call to `LMGetHiliteRGB`.

The for loop is traversed three times. On the second and third traverses, the highlight color is changed.

Within the for loop, a copy of the value at the low memory global `HiliteMode` is retrieved using `LMGetHiliteMode`, `BitClr` is called to clear the highlight bit, and `LMSetHiliteMode` is called to set to low memory global to this new value. At the if/else block, the highlight color is changed if this is the second or third time around the loop. With the highlight bit cleared, `InvertRect` is called to invert a specified rectangle.

Note that the call to `InvertRect` resets the highlight bit. Accordingly, when the user clicks the mouse button, the highlight bit is cleared once again before `InvertRect` is called once again. This second call restores the color in the specified rectangle to the background color.

Before the `doHighLighting` function returns, it sets the highlight color to the saved highlight color.

doDrawWithMouse

`doDrawWithMouse` demonstrates the use of the mouse to define bounding rectangles for QuickDraw shape drawing functions. It also demonstrates the implementation of the "rubber band" rectangle commonly used to provide visual feedback to the user as he drags the mouse during such operations. (While the mouse button remains down, the "rubber-band" rectangle is continually erased and redrawn as the mouse is moved. It is erased when the mouse button is released.)

`doDrawWithMouse` is called when a mouse-down occurs in the window while it is the front window, provided that the global variable `gDrawWithMouseActivated` is set to true.

The call to `GetPixPat` loads a `'ppat'` resource containing a small 8-pixel by 8-pixel pattern. This pixel pattern is assigned to the pen by the call to `PenPixPat`. The call to `PenSize` makes the pen size one pixel high by one pixel wide. The pen pattern mode is then set to `patXOr`. (Note: For a black-and-white "rubber band," replace the `PenPixPat` call with

```
PenPat(GetQDGlobalsGray(&grayPattern));
```

The call to `GetMouse` saves the initial mouse location to a Point variable. The contents of the fields of this variable will remain unchanged. Those coordinates are also used to initialize the left and top fields of the `Rect` variable `drawRect`.

The next call to `GetMouse` assigns the initial location of the mouse to another Point variable. The contents of the fields of this variable will continually change as the mouse is dragged.

The while loop continues to execute while the mouse button remains down. Within the loop, the current mouse location is retrieved and compared with the previous mouse location (the first if statement). If the mouse has moved

- FrameRect is called to draw the framed rectangle.
- If the current mouse horizontal coordinate is greater than or equal to the initial horizontal mouse coordinate, the current mouse horizontal coordinate is assigned to the right field of the rectangle.
- If the current mouse vertical coordinate is greater than or equal to the initial vertical mouse coordinate, the current mouse vertical coordinate is assigned to the bottom field of the rectangle.
- If the current mouse horizontal coordinate is less than or equal to the initial horizontal mouse coordinate, the current mouse horizontal coordinate is assigned to the left field of the rectangle.
- If the current mouse vertical coordinate is less than or equal to the initial vertical mouse coordinate, the current mouse vertical coordinate is assigned to the top field

of the rectangle.
- FrameRect is called again with the newly defined rectangle passed in.

Because the drawing mode is `patXor`, the first call to `FrameRect` erases the old rectangle. Because `FrameRect` is only called if the mouse has moved, the flicker which would otherwise occur when the mouse is stationary is avoided.

Below the if block, and preparatory to the next comparison of current and previous mouse location, the current mouse location becomes the previous mouse location.

When the mouse button is released

- The final call to `FrameRect` erases the final "rubber-band" rectangle.
- The foreground color is set to a random color, the pen pattern mode is set to patCopy, the pen pattern is set to black, and the background pixel pattern is set to that previously used to draw the "rubber band."
- The rectangle as at mouse button release is used in calls to QuickDraw painting and erasing functions to draw rectangles, round rectangles, ovals, and arcs. Just which function is called depends on the value returned by the call to doRandomNumber.
- The background pattern is set to white.

doDrawingState

`doDrawingState` is similar to the function `doDrawListView` in the demonstration program Appearance, the difference being that, in `doDrawingState`, the drawing state is saved at entry and restored at exit.

Note that the call to `NormalizeThemeDrawingState` is included in this function for demonstration purposes only. Ordinarily, this function would be called (if required) at other points in an application.

The call to `GetThemeDrawingState` saves the drawing state prior to the calls to the Appearance Manager functions `SetThemeBackground` and `SetThemePen`, which will change either the color or the pattern settings in the graphics port.

The call to `SetThemeDrawingState` restores the saved drawing state.

The intervening code simply draws a Mac OS 8/9-type list view in the left half of the window.

The calls to do`DrawingStateProof` are also for demonstration purposes only. As will be seen, this function simply draws rectangles in the right half of the window in the pen and background colors and patterns as they were after the call to `NormalizeThemeDrawingState`, after the calls to the Appearance Manager

functions, and after the call to `SetThemeDrawingState`.

doDrawingStateProof

`doDrawingStateProof` is called by `doDrawingState` to draw rectangles in the right half of the window in the pen and background colors and patterns as they were after the call to `NormalizeThemeDrawingState`, after the calls to the Appearance Manager functions, and after the call to `SetThemeDrawingState`.

doRandomNumber

`doRandomNumber` is incidental to the demonstration.

The use of the QuickDraw random number generator is quite adequate for the purposes of this demonstration. However, a professional programmer would not regard it as measuring up to the minimal standards of a "serious" random number generator. (See the article on random number generators at http://www.mactech.com/articles/mactech/vol.08/08.03/randomnumbers/index.html.)

Chapter 13. Offscreen Graphics Worlds, Pictures, Cursors, and Icons

IN THIS CHAPTER

- Offscreen Graphics Worlds
- Pictures
- Cursors
- Icons
- Main Constants, Data Types and Functions-Offscreen Graphics Worlds
- Main Constants, Data Types and Functions-Pictures
- Main Constants, Data Types and Functions-Cursors
- Appearance Manager Constants, Data Types and Functions-Cursors
- Main Constants, Data Types and Functions-Icons
- Demonstration Program GworldPicCursIcn Listing
- Demonstration Program GWorldPicCursIcn Comments

Offscreen Graphics Worlds

Introduction

An **offscreen graphics world** may be regarded as a virtual screen on which your application can draw a complex image without the user seeing the various steps involved. When your application draws into an offscreen graphics world, it draws into a part of memory not used by the video device. Thus the drawing process remains hidden from the user. When the drawing is completed, your application can copy the image from the offscreen graphics world to the active window using the `CopyBits`, `CopyMask`, or `CopyDeepMask` functions.

One of the key advantages of using an offscreen graphics world is speed. Copying a complex image from an offscreen graphics world to the active window is much faster than performing all the steps necessary to draw the image on-screen.

Creating an Offscreen Graphics World

The `NewGWorld` function is used to create an offscreen graphics world:

```
QDErr  NewGWorld(GWorldPtr *offscreenGWorld,short PixelDepth,const Rect *boundsRect,
                 CTabHandle cTable,GDHandle aGDevice,GWorldFlags flags)
```

Returns: A result code: `noErr` (no error); `paramErr` (illegal parameter); `cDepthErr` (invalid pixel depth).

`offscreenGWorld` Pointer to the created offscreen graphics world.

`PixelDepth` Pixel depth of the offscreen graphics world. Possible depths are 0, 1, 2, 4, 8, 16, and 32 bits per pixel. Specifying `0` sets the pixel depth to equal the greatest depth of those screens whose boundary rectangles intersect the rectangle passed in the `boundRect` parameter. `0` also causes `NewGWorld` to use the `GDevice` structure for this deepest device rather than create a new one.

`boundsRect` The offscreen pixel map's boundary and port rectangle. Applications typically pass in the port rectangle of the window to which the image in the offscreen graphics world will be copied.

`cTable` Handle to a `ColorTable` structure. May be `NULL`.

`aGDevice` Handle to a `GDevice` structure. This is used only when `noNewDevice` is passed in the `flags` parameter. `NewGWorld` will attach this `GDevice` structure to the offscreen graphics world. Should be `NULL` if `0` is passed in the `PixelDepth` parameter.

`flags` Any combination of `pixPurge` (make base address of pixel image purgeable), `noNewDevice` (do not create offscreen `GDevice` structure), `useTempMem` (create base address for `offscreen` pixel image in temporary memory, and `keepLocal` (keep offscreen pixel image in main memory) may be passed in this parameter.

Calling `NewGWorld` results in the creation of a new offscreen graphics port. The function returns, in the `offscreenGWorld` parameter, a pointer of type `GWorldPtr` which points to the graphics port:

```
typedef CGrafPtr GWorldPtr;
```

`NewGWorld` also establishes a link to an existing `GDevice` structure, or creates a new `GDevice` structure and establishes a link to that.

Passing `0` in the `PixelDepth` parameter, a window's port rectangle in the `boundsRect` parameter, `NULL` in both the `cTable` and `aGDevice` parameters, and `0` in the `flags` parameter:

- Allows QuickDraw to optimize the `CopyBits`, `CopyMask`, and `CopyDeepMask` functions used to copy the image into the window's port rectangle.
- Results in the default behavior of `NewGWorld`, meaning that the base address of the offscreen pixel image is unpurgeable, memory in the application heap is used, and graphics accelerators can cache the offscreen pixel image.

Setting the Graphics Port

Before drawing into the offscreen graphics port, you should save the current graphics port and the current device's `GDevice` structure by calling `GetGWorld`. The offscreen graphics port should then be made the current port by a call to `SetGWorld`. After drawing into the offscreen graphics world, you should call `SetGWorld` to restore the saved graphics port as the current graphics port.

`SetGWorld` takes two parameters (`port` and `gdh`). If the `port` parameter is of type `CGrafPtr`, the current port is set to the port specified in the `port` parameter and the current device is set to the device specified in the `gdh` parameter. If the `port` parameter is of type `GWorldPtr`, the current port is set to the port specified in the `port` parameter, the `gdh` parameter is ignored, and the current device is set to the device linked to the offscreen graphics world.

Preparing to Draw into an Offscreen Graphics World

After setting the offscreen graphics world as the current port, you should use the `GetGWorldPixMap` function to get a handle to the offscreen pixel map. This is required as the parameter in a call to the `LockPixels` function, which you must call before drawing to, or copying from, an offscreen graphics world.

`LockPixels` prevents the base address of an offscreen pixel image from being moved while you draw into it or copy from it. It returns `true` if the base address is not purgeable, or if the base address has not been purged by the Memory Manager. If `LockPixels` returns `false`, (meaning that the base address of the offscreen pixel image has been purged) your application must call the `UpdateGWorld` function to reallocate the offscreen pixel image and then reconstruct it.

As a related matter, note that the `baseAddr` field of the `PixMap` structure for an offscreen graphics world contains a handle, whereas the `baseAddr` field for an onscreen pixel map contains a pointer. Accordingly, the `GetPixBaseAddr` function must be used to obtain a pointer to the `PixMap` structure for an offscreen graphics world.

Copying an Offscreen Image into a Window

After drawing the image in the offscreen graphics world, your application should call `SetGWorld` to set the active window as the current graphics port preparatory to copying the image to that port.

Your application copies the image from the offscreen graphics world into the target window using `CopyBits` (or, if masking is required, `CopyMask` or `CopyDeepMask`). Note that `CopyBits`, `CopyMask` and `CopyDeepMask` expect their source and destination parameters to be pointers to bit maps, not pixel maps. (These functions date from the era of black-and-white Macintoshes, which is why they expect a pointer to a bitmap. By looking at certain information in the graphics ports, `CopyBits`, `CopyMask`, and `CopyDeepMask` can establish that you have passed the functions a handle to a pixel map rather than the base address of a bitmap.)

You must leave the pixel image locked while you are drawing into an offscreen graphics world or copying an image from it, and you should call `UnlockPixels` when you are finishing the copying or drawing operation. (Calling `UnlockPixels` will assist in preventing heap fragmentation.)

Updating an Offscreen Graphics World

If, for example, you are using an offscreen graphics world to support the window updating process, you can use `UpdateGWorld` to carry certain changes affecting the window (resizing the window, changes to the pixel depth of the screen, and so on) through to the offscreen graphics world. Calling `UpdateGWorld` obviates the necessity to re-create the offscreen graphics world and redraw its contents.

Disposing of an Offscreen Graphics World

You should call `DisposeGWorld` when your application no longer needs the offscreen graphics world.

Pictures

Introduction

QuickDraw provides a set of functions that allow your application to record a number of drawing commands and subsequently play the recording back. The collection of drawing commands is called a **picture.**

You begin defining a picture by calling the function `OpenCPicture`. Your subsequent drawing commands are collected in a data structure of type `Picture`. The picture defined within this data structure may be drawn by calling the function `DrawPicture`.

The `OpenCPicture` function creates pictures in the **extended version 2 format,** which allows your application to specify resolutions for pictures.

The Picture Structure

The `Picture` structure is as follows:

```
struct Picture
{
  short picSize;
  Rect  picFrame;
} ;
typedef struct Picture Picture;
typedef Picture *PicPtr;
typedef PicPtr *PicHandle;
```

Field Descriptions

`picSize` This field is irrelevant for version 2 format and extended version 2 format pictures.

> **Note**
>
> To determine the size of a picture in memory, use the Memory Manager function `GetHandleSize`. To determine the size of a picture in a file of type `'PICT'`, use the File Manager function `PBGetFInfo`. To determine the size of a picture in a resource of type `'PICT'`, use the Resource Manager function `MaxSizeResource`.

`picFrame` The picture's bounding rectangle. When you draw into a differently sized rectangle, `DrawPicture` uses this rectangle to scale the picture.

`...` Compact drawing commands and picture comments constitute the rest of the structure, which is of variable length.

Opcodes: Drawing Commands and Picture Comments

The variable length field in a `Picture` structure contains data in the form of **opcodes,** which `DrawPicture` uses to determine what objects to draw or what mode to change for subsequent drawing. Opcodes can also specify **picture comments,** which are created using `PicComment`. A picture comment contains data or commands for special processing by output devices, such as PostScript printers.

You typically use QuickDraw commands when drawing to the screen and picture comments to include any special drawing commands for printers.

'PICT' Files, Resources, and Scrap Format

File Manager and Resource Manager functions are used to read pictures from, and write pictures to, a disk. Scrap Manager functions are used to read pictures from, and write pictures to, the scrap (see Chapter 20).

A picture can be stored as a `'PICT'` resource in the resource fork of any file type. A picture can also be stored in the data fork of a file of type `'PICT'`. The first 512 bytes of the data fork of a `'PICT'` file are a header that your application can use for its own purposes.

The Scrap Manager maintains a storage area to hold the last data cut or copied by the user. This area is called the **scrap.** If your application supports cut, copy, and paste operations, it necessarily reads data from, and writes data to, the scrap. There are two standard scrap data formats, one of which is `'PICT'`.

Creating Pictures

As previously stated, you use the `OpenCPicture` function to begin defining a picture. You pass information to `OpenCPicture` in the form of an `OpenCPicParams` structure:

```
struct OpenCPicParams
{
  Rect  srcRect;    // Optimal bounding rectangle.
```

```
  Fixed hRes;       // Best horizontal resolution.
  Fixed vRes;       // Best vertical resolution.
  short version;    // Set to -2.
  short reserved1;  // (Reserved. Set to 0.)
  long  reserved2;  // (Reserved. Set to 0.)
} ;
typedef struct OpenCPicParams OpenCPicParams;
```

This structure provides a simple mechanism for specifying resolutions when creating images. For example, applications that create pictures from scanned images can specify resolutions higher than 72 dpi.

You call `ClosePicture` to complete the collection of drawing (and picture comment) commands that define your picture.

Clipping Region

Before calling `OpenCPicture`, you should always use `ClipRect` to specify an appropriate clipping region. If you fail to do this, `OpenCPicture` will use the clipping region contained in the current graphics port object. By default, this region is very large (the size of the coordinate plane). In this circumstance, if you scale the picture when drawing it, the clipping region can become invalid and your picture will not be drawn. By the same token, if your application has previously set the clipping region for some other purpose, part of your drawing may be clipped.

Ordinarily, you should set the clipping region to equal the port rectangle of the current graphics port before recording a picture.

Opening and Drawing Pictures

You can retrieve pictures saved in `'PICT'` files using File Manager functions. You can retrieve pictures saved in the resource forks of other file types using the `GetPicture` function. You can retrieve pictures stored in the scrap using the Carbon Scrap Manager function `GetScrapFlavorData`.

> **Note**
>
> The demonstration program in Chapter 18 shows how to read pictures from, and save pictures to, files of type `'PICT'`.

When the picture is retrieved, you can call `DrawPicture` to draw the picture. The second parameter passed in the `DrawPicture` function is the destination rectangle, which should

be specified in coordinates local to the current graphics port. `DrawPicture` shrinks or stretches the picture as necessary to make it fit into this rectangle.

When you are finished using a picture stored as a `'PICT'` resource, you should use the resource Manager function `ReleaseResource` to release its memory.

Saving Pictures

To save a picture in a `'PICT'` file, you should use the appropriate File Manager functions. (Remember that the first 512 bytes of a `'PICT'` file are reserved for your application's own purposes.) To save pictures in a `'PICT'` resource, you should use the appropriate Resource Manager functions. To place a picture in the scrap (for example, to respond to the user choosing the Copy command to copy a picture to the clipboard), you should use the Carbon Scrap Manager function `PutScrapFlavorData`.

Gathering Picture Information

`GetPictInfo` may be used to gather information about a single picture, and `GetPixMapInfo` may be used to gather color information about a single pixel map or bit map. Each of these functions returns color and resolution information in a `PictInfo` structure. A `PictInfo` structure can also contain information about the drawing objects, fonts, and comments in a picture.

Cursors

Introduction

A **cursor** is a 16-by-16 pixel image defined in a black-and-white cursor (`'CURS'`) or color cursor (`'crsr'`) resource.

Cursor Movement, Hot Spot, Visibility, and Shape

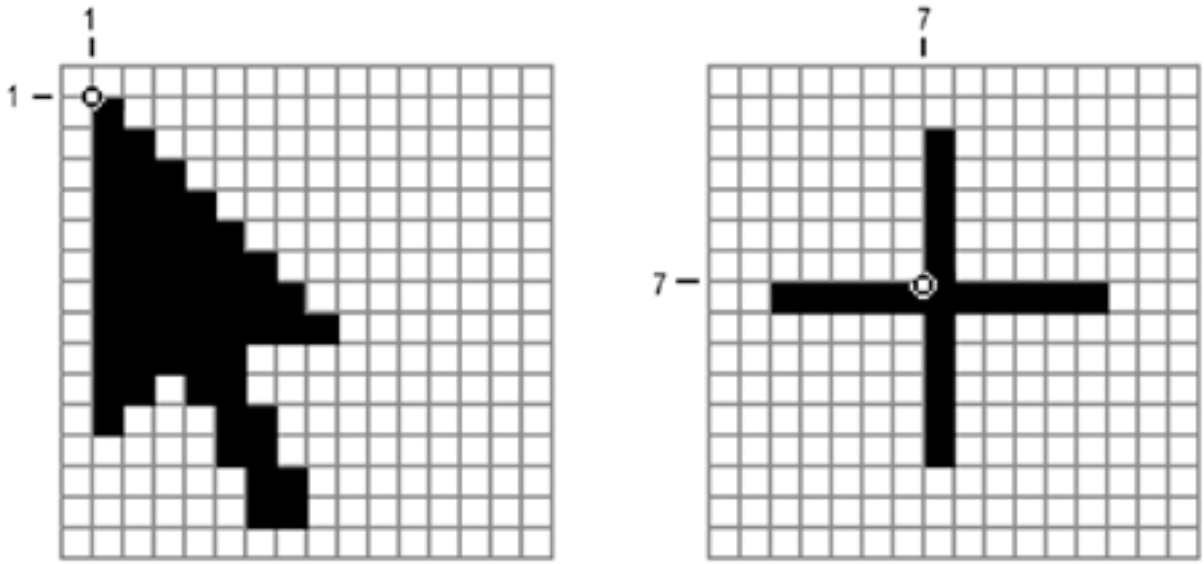
Cursor Movement

Cursor movement is not the responsibility of your application. When the mouse is moved by the user, low-level interrupt-driven mouse functions move the cursor on the screen.

Cursor Hot Spot

A cursor's **hot spot** is that part of the cursor that actually points to an object on the screen. Mouse clicks only have an effect on that object when the hot spot, not the cursor as a whole, is over the object. Fig 1 illustrates two cursors and their hot spot points. Note that the hot spot is a point, not a bit.

Figure 1. CURSOR HOT SPOTS



Cursor Visibility

Generally speaking, your application should always make the cursor visible. There are,

however, exceptions to this rule. For example, in a text-editing application, the cursor should be made invisible, and the insertion point made to blink, when the user begins entering text. In such cases, the cursor should be made visible again only when the user moves the mouse.

Cursor Shape

Your application should change the shape of the cursor in the following circumstances:

- To indicate that the user is over a certain area of the screen. When the cursor is in the menu bar, for example, it should usually have an arrow shape. When the user moves the cursor over a text document, the cursor shape should be changed to the I-beam shape.
- To provide feedback to the user indicating that a time-consuming operation is in progress. For example, if an operation will take a second or two, you should provide feedback to the user by changing the cursor to the wristwatch cursor (on Mac OS 8/9) or wait cursor (Mac OS X) (see Fig 2). If the operation will take several seconds and the only options available to the user are to stop the operation, wait until it is completed, or switch to another application, you should display an animated cursor (on Mac OS 8/9) or wait cursor (on Mac OS X).

Figure 2. THE I-BEAM, CROSSHAIRS, PLUS SIGN, AND WRISTWATCH CURSORS



Note

If the operation takes longer than several seconds, you should display a dialog with a progress indicator (see Chapter 25).

Non-Animated Cursors

System 'CURS' and 'crsr' Resources

The system contains a number of `'CURS'` and `'crsr'` resources. The following constants represent the `'CURS'` resource IDs for the basic cursors shown in Fig 2:

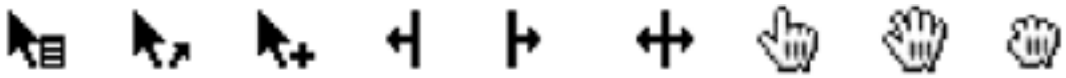
Constant	*Value*	*Description*
`iBeamCursor`	`1`	Used in text editing

`crossCursor`	`2`	Often used for manipulating graphics
`plusCursor`	`3`	Often used for selecting fields in an array
`watchCursor`	`4`	Used when a short operation is in progress

The following lists the `'CURS'` and `'crsr'` resource IDs for the additional cursors shown in Fig 3:

Constant	*Value*	*Description*
-	`-20488`	Contextual menu arrow cursor
-	`-20487`	Alias arrow cursor
-	`-20486`	Copy arrow cursor
-	`-20452`	Resize left cursor. (Not available on Mac OS X.)
-	`-20451`	Resize right cursor. (Not available on Mac OS X.)
-	`-20450`	Resize left/right cursor. (Not available on Mac OS X.)
-	`-20877`	Pointing hand cursor. (Not available on Mac OS X.)
-	`-20876`	Open hand pointer. (Not available on Mac OS X.)
-	`-20875`	Close hand pointer. (Not available on Mac OS X.)

Figure 3. ADDITIONAL CURSOR AND COLOUR CURSOR RESOURCES



Custom 'CURS' and 'crsr' Resources

To create custom cursors, you need to define `'CURS'` or `'crsr'` resources in the resource file of your application.

Changing Cursor Shape

Your application is responsible for setting the initial appearance of the cursor and for changing the appearance of the cursor as appropriate for your application.

Methodology 1

One method for changing cursor shape involves first getting a handle to the relevant cursor

(either a custom cursor or one of the system cursors shown in Figs 2 and 3) by specifying its resource ID in a call to `GetCursor` or `GetCCursor`. `GetCursor` returns a handle to a `Cursor` structure. `GetCCursor` returns a handle to a `CCrsr` structure. The address of the `Cursor` or `CCrsr` structure is then used in a call to `SetCursor` or `SetCCursor` to change the cursor shape.

Methodology 2

Mac OS 8.5 introduced a new method for setting the cursor. You must pass one of the following constants, which are of type `ThemeCursor`, in the `inCursor` parameter of the function `SetThemeCursor`:

Constant	*Value*	*Comments*
`kThemeArrowCursor`	`0`	
`kThemeCopyArrowCursor`	`1`	
`kThemeAliasArrowCursor`	`2`	
`kThemeContextualMenuArrowCursor`	`3`	
`kThemeIBeamCursor`	`4`	
`kThemeCrossCursor`	`5`	
`kThemePlusCursor`	`6`	
`kThemeWatchCursor`	`7`	Can animate
`kThemeClosedHandCursor`	`8`	
`kThemeOpenHandCursor`	`9`	
`kThemePointingHandCursor`	`10`	
`kThemeCountingUpHandCursor`	`11`	Can animate
`kThemeCountingDownHandCursor`	`12`	Can animate
`kThemeCountingUpAndDownHandCursor`	`13`	Can animate
`kThemeSpinningCursor`	`14`	Can animate
`kThemeResizeLeftCursor`	`15`	
`kThemeResizeRightCursor`	`16`	
`kThemeResizeLeftRightCursor`	`17`	

Changing Cursor Shape in Response to Mouse-Moved Events

Most applications set the cursor to the I-beam shape when the cursor is inside a text-editing area of a document, and they change the cursor to an arrow when the cursor is inside the scroll bars. Your application can achieve this effect by requesting that the Event Manager report mouse-moved events if the user moves the cursor out of a region you specify in the `mouseRgn` parameter to the `WaitNextEvent` function. Then, when a mouse-moved event is detected in your main event loop, you can use `SetCursor`, `SetCCursor`, or `SetThemeCursor`, to change the cursor to the appropriate shape.

Changing Cursor Shape in Response to Resume Events

Your application also needs to set the cursor shape in response to resume events, normally by setting the arrow cursor.

Hiding Cursors

You can remove the cursor image from the screen using `HideCursor`. You can hide the cursor temporarily using `ObscureCursor` or you can hide the cursor in a given rectangle by using `ShieldCursor`. To display a hidden cursor, use `ShowCursor`. Note, however, that you do not need to explicitly show the cursor after your application uses `ObscureCursor` because the cursor automatically reappears when the user moves the mouse again.

Animated Cursors-Mac OS 8/9

Methodology 1

Mac OS 8.5 introduced a new function (`SetThemeAnimatedCursor`) for animating a specified cursor type. You must pass one of the following constants, which are of type `ThemeCursor`, in the `inCursor` parameter of `SetThemeAnimatedCursor`:

Constant	*Value*
`kThemeWatchCursor`	`7`
`kThemeCountingUpHandCursor`	`11`
`kThemeCountingDownHandCursor`	`12`
`kThemeCountingUpAndDownHandCursor`	`13`
`kThemeSpinningCursor`	`14`

Methodology 2

Another methodology requires

- A series of `'CURS'` (or `'crsr'`) resources that make up the "frames" of the animation.
- An `'acur'` resource, which collects and orders the `'CURS'` frames into a single animation, specifying the IDs of the resources and the sequence for displaying them in the animation.

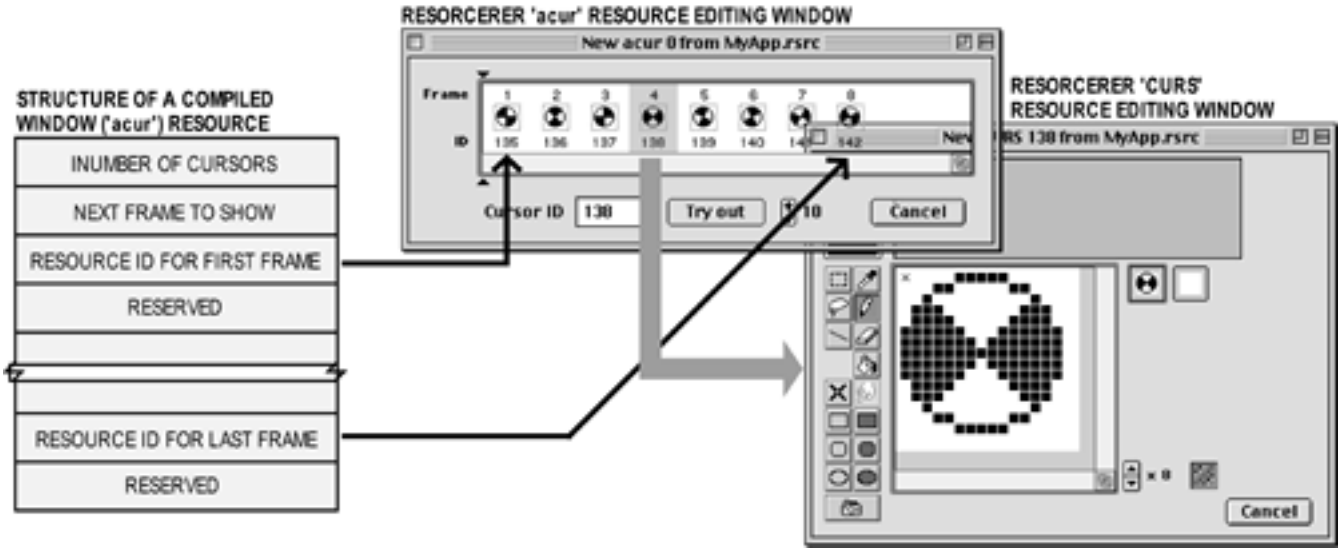
System 'acur' and 'CURS' Resources

The system contains an `'acur'` resource (ID -6079), together the associated eight `'CURS'` resources, for an animated watch cursor. It also contains eight `'CURS'` resources (IDs -20701 to -20708) for an animated spinning (beach ball) cursor and six `'CURS'` resources (IDs -20709 to -20714) for an animated counting hand cursor.

Custom 'acur' and 'CURS' Resources

Fig 4 shows the structure of a compiled `'acur'` resource, and an `'acur'` resource and one of its associated `'CURS'` resources being created using Resorcerer.

Figure 4. CREATING AN 'acur' RESOURCE AND ASSOCIATED 'CURS' RESOURCES USING RESORCERER



Creating the Animated Cursor

The following are the steps required to create the animated cursor:

- If you do not intend to use the system-supplied `'acur'` and associated `'CURS'` resources
    - Create a series of `'CURS'` resources that make up the "frames" of the animation

    - Create an `'acur'` resource

- Load the `'acur'` resource into a structure which replicates the structure of an `'acur'` resource, for example

```
typedef struct
{
  short      numberOfFrames;
  short      whichFrame;
  CursHandle frame[];
} animCurs, *animCursPtr, **animCursHandle;
```

- Load the `'CURS'` resources using `GetCursor` and assign handles to the resulting `Cursor` structures to the elements of the `frame` field.

- At the desired interval, call `SetCursor` to display each cursor, that is, each "frame," in rapid succession, returning to the first frame after the last frame has been displayed.

Animated Cursor-Mac OS X

When the Mac OS X wait cursor appears automatically, it means that the application has stopped calling an event handling API for more than a certain period of time (about two seconds).

Your application can also turn the wait cursor on and off using the QuickDraw function `QDDisplayWaitCursor`. Passing `true` in the `forceWaitCursor` parameter turns the cursor on and passing `false` resumes automatic operation. The function keeps track of nested calls.

Icons

Icons and the Finder-Icon Families

As stated in Chapter 9, the Finder uses **icons** to graphically represent objects, such as files and directories. Chapter 9 also introduced the subject of **icon families,** and stated that your application should provide the Finder with a family of specially designed icons for the application file itself and for each of the document types created by the application.

Other Icons-Icons, Color Icons and Small Icons

Other icon types are the **icon, color icon,** and **small icon.** Note that the Finder does not use or display these icon types.

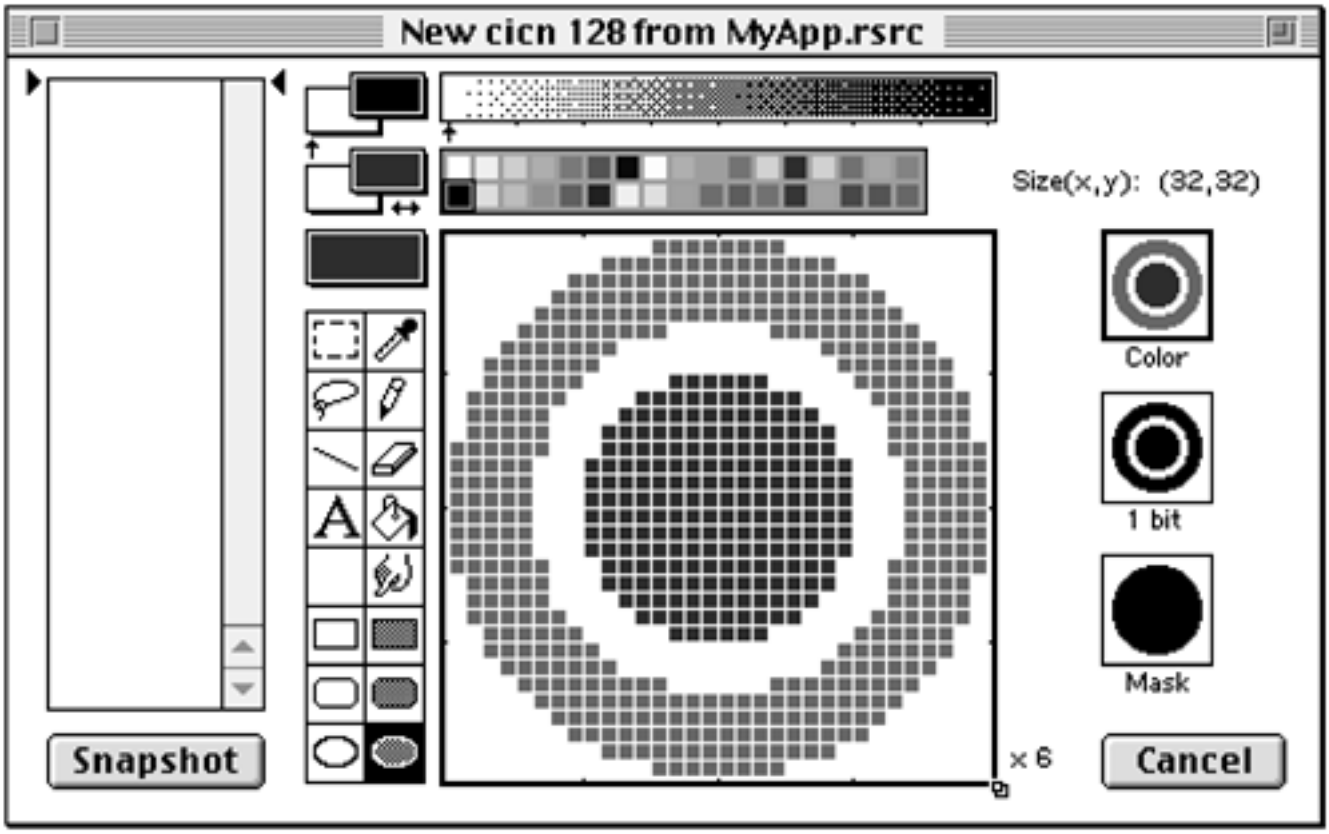
Icon ('ICON')

The icon is a black-and-white icon defined in an `'ICON'` resource, which contains a 32-by-32 pixel bit map. Icons do not need a mask because they are always displayed on a white background.

Color Icon ('cicn')

The color icon is defined in a `'cicn'` resource, which includes a pixel map, a bit map, and a mask. You can use a `'cicn'` resource to define a color icon with any width and height and with a bit depth up to 8. Fig 5 shows an 8-bit 32-by 32-pixel `'cicn'` resource being created using Resorcerer.

Figure 5. CREATING AN 8-BIT 32 BY 32 PIXEL 'cicn' RESOURCE USING RESORCERER



Small Icon ('SICN')

The small icon is a black-and-white icon defined in a `'SICN'` resource. Small icons are 12 by 16 pixels even though they are stored in a resource as 16-by-16 pixel bitmaps. Small icons are of doubtful utility in the Carbon era and will not be considered further.

Icons in Windows, Menus, and Alerts and Dialogs

The icons provided by your application for the Finder (or the default system-supplied icons used by the Finder if your application does not provide its own icons) are displayed on the desktop. Your application can also display icons in its menus, dialogs and windows.

Icons in Windows

You can display icons of any kind in your windows using the appropriate Icon Utilities functions.

Icons in Menus

The Menu Manager allows you to display icons of resource types `'ICON'` (icon) and `'cicn'` (color icon) in menu items. The procedure is as follows:

- Create the icon resource with a resource ID between 257 and 511. Subtract 256 from

the resource ID to get a value called the **icon number.** Specify the icon number in the Icon field of the menu item definition.

- For an icon (`'ICON'`), specify `0x1D` in the keyboard equivalent field of the menu item definition to indicate to the Menu Manager that the icon should be reduced to fit into a 16-by-16 pixel rectangle. Otherwise, specify a value of `0x00`, or a value greater than `0x20`, in the keyboard equivalent field to cause the Menu Manager to expand the item's rectangle so as to display the icon at its normal 32-by-32 pixel size. (A value greater than `0x20` in the keyboard equivalent field specifies the item's Command-key equivalent.)

- For a color icon (`'cicn'`), specify `0x00` or a value greater than `0x20` in the keyboard equivalent field of the menu item definition. The Menu Manager automatically enlarges the enclosing rectangle of the menu item according to the rectangle specified in the `'cicn'` resource. (Color icons, unlike icons, can be any height or width.)

When the menu is displayed, the Menu Manager first looks for a `'cicn'` resource with the resource ID calculated from the icon number and displays that icon if it is found. If a `'cicn'` resource is not found, the Menu Manager searches for an `'ICON'` resource and plots it in either a 32-by-32 pixel rectangle or a 16-by-16 bit rectangle, depending on the value in the menu item's keyboard equivalent field.

Icons in Alerts and Dialogs

The Dialog Manager allows you to display icons of resource types `'ICON'` (icon) and `'cicn'` (color icon) in Mac OS 8/9 alerts and in dialogs. You can display the icon alone or within an image well.

To display the icon alone, the procedure is to define an item of type `Icon` and provide the resource ID of the icon in the item list (`'DITL'`) resource for the dialog. This will cause the Dialog Manager to automatically display the icon whenever you display the alert or dialog using Dialog Manager functions.

To display the icon within an image well, include an image well control in the alert or dialog's item list and assign the resource ID of the icon to the control's minimum value field.

If you provide a color icon (`'cicn'`) resource with the same resource ID as an icon (`'ICON'`) resource, the Dialog Manager displays the color icon instead of the black-and-white icon.

On Mac OS 8/9, you would ordinarily use the `Alert` function (which does not automatically draw a system-supplied alert icon in the alert), or the `StandardAlert` function with `kAlertPlainAlert` passed in the `inAlertType` parameter, when you wish to display an alert containing your own icon (for example, in your application's About… alert). If you invoke an alert using the `NoteAlert`, `CautionAlert`, or `StopAlert` functions, or with the

`StandardAlert` function with an alert type constant of other than `kAlertPlainAlert` passed in the `inAlertType` parameter, the Dialog Manager draws the system-supplied icon as well as your icon. Since your icon is drawn last, you can obscure the system-supplied icon by positioning your icon at the same coordinates.

Drawing and Manipulating Icons

The Icon Utilities allow your application (and the system software) to draw and manipulate icons of any standard resource type in windows and, subject to the limitations and requirements previously described, in menus and dialogs.

You need to use Icon Utilities functions only if:

- You wish to draw icons in your application's windows.
- You wish to draw icons which are not recognized by the Menu Manager and the Dialog Manager in, respectively, menu items and dialogs.

Preamble-Icon Families and Icon Suites

Icon Families

You can define individual icons of resource types `'ICON'` and `'cicn'` that are not part of an icon family and use Icon Utilities functions to draw them as required. However, to display an icon effectively at a variety of sizes and bit depths, you should provide an icon family in the same way that you provide icon families for the Finder. The advantage of providing an icon family is that you can then leave it to functions such as `PlotIconID`, which are used to draw icons, to automatically determine which icon in the icon family is best suited to the specified destination rectangle and current display bit depth.

Icon Suites

Some Icon Utilities functions take as a parameter a handle to an **icon suite.** Typically, an icon suite is comprised of one or more handles to icon resources from a single icon family which have been read into memory. The `GetIconSuite` function may be used to get a handle to an icon suite, which can then be passed to functions such as `PlotIconSuite` to draw that icon in the icon suite best suited to the destination rectangle and current display bit depth.

An icon suite can contain handles to all of the six icon resources that an icon family can contain. Alternatively, it can contain handles to only a subset of those resources.

When you create an icon suite from icon family resources, the associated resource file should remain open while you use Icon Utilities functions.

Drawing an Icon Directly from a Resource

To draw an icon from an icon family without first creating an icon suite, use the `PlotIconID` function. `PlotIconID` determines, from the size of the specified destination rectangle and the current bit depth of the display device, which icon to draw. The icon drawn is as follows:

Destination Rectangle Size	*Icon Drawn*
Width or height greater than or equal to 32	The 32-by-32 pixel icon with the appropriate bit depth
Less than 32 by 32 pixels bit depth	The 16-by-16 pixel icon with the appropriate

Icon Stretching and Shrinking

`PlotIconID` may stretch or shrink the icon to fit depending on the size of the destination rectangle. To draw icons without stretching them, `PlotIconID` requires that the destination rectangle have the same dimensions as one of the standard icons.

Icon Alignment and Transform

In addition to destination rectangle and resource ID parameters, `PlotIconID` takes **alignment** and **transform** parameters. Icon Utilities functions can automatically align an icon within its destination rectangle. (For example, an icon which is taller than it is wide can be aligned to either the right or left of its destination rectangle.) These functions can also transform the appearance of the icon in standard ways analogous to Finder states for icons.

Variables of type `IconAlignmentType` and `IconTransformType` should be declared and assigned values representing alignment and transform requirements. Constants, such as `kAlignAbsoluteCenter` and `kTransformNone`, are available to specify alignment and transform requirements.

Getting an Icon Suite and Drawing One of Its Icons

The `GetIconSuite` function, with the constant `kSelectorAllAvailableData` passed in the third parameter, is used to get all icons from an icon family with a specified resource ID and to collect the handles to the data for each icon into an icon suite. An icon from this suite may then be drawn using `PlotIconSuite` which, like `PlotIconID`, takes destination rectangle, alignment and transform parameters and stretches or shrinks the icon if necessary.

Drawing Specific Icons from an Icon Family

If you need to plot a specific icon from an icon family rather than use the Icon Utilities to automatically select a family member, you must first create an icon suite that contains only the icon of the desired resource type together with its corresponding mask. Constants such as

`kSelectorLarge4Bit` (an icon selector mask for an `'icl4'` icon) are used as the third parameter of the `GetIconSuite` call to retrieve the required family member. You can then use `PlotIconSuite` to plot the icon.

Drawing Icons That Are Not Part of an Icon Family

To draw icons of resource type `'ICON'` and `'cicn'` in menu items and dialogs, you use Menu Manager and Dialog Manager functions such as `SetItemIcon` and `SetDialogItem`.

To draw resources of resource type `'ICON'` and `'cicn'` in your application's windows, you use the following functions:

Resource Type	*Function to Get Icon*	*Functions to Draw Icon*
`'ICON'`	`GetIcon`	`PlotIconHandle` `PlotIcon`
`'cicn'`	`GetCIcon`	`PlotCIconHandle` `PlotCIcon`

The functions in this list ending in `Handle` allow you to specify alignment and transforms for the icon.

Manipulating Icons

The `GetIconFromSuite` function may be used to get a handle to the pixel data for a specific icon from an icon suite. You can then use this handle to manipulate the icon data, for example, to alter its color or add three-dimensional shading.

The Icon Utilities also include functions that allow you to perform an action on one or more icons in an icon suite and to perform hit testing on icons.

Main Constants, Data Types and Functions-Offscreen Graphics Worlds

Constants

Flags for GWorldFlags Parameter

```
pixPurgeBit        = 0  Set to make base address for offscreen pixel image purgeable.
noNewDeviceBit     = 1  Set to not create a new GDevice structure for offscreen world.
pixelsPurgeableBit = 6  Set to make base address for pixel image purgeable.
pixelsLockedBit    = 7  Set to lock base address for offscreen pixel image.
```

Data Types

```
typedef CGrafPtr       GWorldPtr;
typedef unsigned long  GWorldFlags;
```

Functions

Creating, Altering, and Disposing of Offscreen Graphics Worlds

```
QDErr         NewGWorld(GWorldPtr *offscreenGWorld,short PixelDepth,
              const Rect *boundsRect,CTabHandle cTable,GDHandle aGDevice,GWorldFlags flags);
GWorldFlags   UpdateGWorld(GWorldPtr *offscreenGWorld,short pixelDepth,
              const Rect *boundsRect,CTabHandle cTable,GDHandle aGDevice,GWorldFlags flags);
void          DisposeGWorld(GWorldPtr offscreenGWorld);
```

Saving and Restoring Graphics Ports and Offscreen Graphics Worlds

```
void          GetGWorld(CGrafPtr *port,GDHandle *gdh);
void          SetGWorld(CGrafPtr port,GDHandle gdh);
```

Managing an Offscreen Graphics World's Pixel Image

```
PixMapHandle  GetGWorldPixMap(GWorldPtr offscreenGWorld);
Boolean       LockPixels(PixMapHandle pm);
void          UnlockPixels(PixMapHandle pm);
void          AllowPurgePixels(PixMapHandle pm);
void          NoPurgePixels(PixMapHandle pm);
GWorldFlags   GetPixelsState(PixMapHandle pm);
void          SetPixelsState(PixMapHandle pm,GWorldFlags state);
Ptr           GetPixBaseAddr(PixMapHandle pm);
Boolean       PixMap32Bit(PixMapHandle pmHandle);
```

Main Constants, Data Types and Functions-Pictures

Constants

Verbs for the GetPictInfo, GetPixMapInfo, and NewPictInfo calls

```
returnColorTable      = 0x0001  Return a ColorTable structure.
returnPalette         = 0x0002  Return a Palette structure.
recordComments        = 0x0004  Return comment information.
recordFontInfo        = 0x0008  Return font information.
suppressBlackAndWhite = 0x0010  Do not include black and white.
```

Color Pick Methods for the GetPictInfo, GetPixMapInfo, and NewPictInfo calls

```
systemMethod          = 0       System color pick method.
popularMethod         = 1       Most popular set of colors.
medianMethod          = 2       A good average mix of colors.
```

Data Types

Picture

```
struct Picture
{
  short picSize;    // For a version 1 picture: its size.
  Rect  picFrame;   // Bounding rectangle for the picture
} ;
typedef struct Picture Picture;
typedef Picture *PicPtr;
typedef PicPtr *PicHandle;
```

OpenCPicParams

```
struct OpenCPicParams
{
  Rect  srcRect;    // Optimal bounding rectangle.
  Fixed hRes;       // Best horizontal resolution.
  Fixed vRes;       // Best vertical resolution.
  short version;    // Set to -2
  short reserved1;  // (Reserved. Set to 0.)
  long  reserved2;  // (Reserved. Set to 0.)
} ;
typedef struct OpenCPicParams OpenCPicParams;
```

PictInfo

```
struct  PictInfo
{
  short             version;          // This is always zero, for now.
  long              uniqueColors;     // Number of actual colors in the picture(s)/pixmap
(s).
  PaletteHandle     thePalette;       // Handle to the palette information.
  CTabHandle        theColorTable;    // Handle to the color table.
  Fixed             hRes;             // Maximum horizontal resolution for all the pixmaps.
  Fixed             vRes;             // Maximum vertical resolution for all the pixmaps.
  short             depth;            // Maximum depth for all the pixmaps (in the picture).
  Rect              sourceRect;       // Picture frame rectangle (contains the entire
picture).
  long              textCount;        // Total number of text strings in the picture.
```

```
  long              lineCount;        // Total number of lines in the picture.
  long              rectCount;        // Total number of rectangles in the picture.
  long              rRectCount;       // Total number of round rectangles in the picture.
  long              ovalCount;        // Total number of ovals in the picture.
  long              arcCount;         // Total number of arcs in the picture.
  long              polyCount;        // Total number of polygons in the picture.
  long              regionCount;      // Total number of regions in the picture.
  long              bitMapCount;      // Total number of bitmaps in the picture.
  long              pixMapCount;      // Total number of pixmaps in the picture.
  long              commentCount;     // Total number of comments in the picture.
  long              uniqueComments;   // The number of unique comments in the picture.
  CommentSpecHandle commentHandle;    // Handle to all the comment information.
  long              uniqueFonts;      // The number of unique fonts in the picture.
  FontSpecHandle    fontHandle;       // Handle to the FontSpec information.
  Handle            fontNamesHandle;  // Handle to the font names.
  long  reserved1;
  long  reserved2;
} ;
typedef struct PictInfo PictInfo;
typedef PictInfo *PictInfoPtr;
typedef PictInfoPtr *PictInfoHandle;
```

CommentSpec

```
struct CommentSpec
{
  short count;       // Number of occurrences of this comment ID.
  short ID;          // ID for the comment in the picture.
} ;
typedef struct CommentSpec CommentSpec;
typedef CommentSpec *CommentSpecPtr;
typedef CommentSpecPtr *CommentSpecHandle;
```

FontSpec

```
struct FontSpec
{
  short pictFontID;  // ID of the font in the picture.
  short sysFontID;   // ID of the same font in the current system file.
  long  size[4];     // Bit array of all the sizes found (1..127) (bit 0 means > 127).
  short style;       // Combined style of all occurrances of the font.
  long  nameOffset;  // Offset into the fontNamesHdl handle for the font's name.
} ;
typedef struct FontSpec FontSpec;
typedef FontSpec *FontSpecPtr;
typedef FontSpecPtr *FontSpecHandle;
```

Functions

Creating and Disposing of Pictures

```
PicHandle  OpenCPicture(const OpenCPicParams *newHeader);
void       PicComment(short kind,short dataSize,Handle dataHandle);
void       ClosePicture(void);
void       KillPicture(PicHandle myPicture);
```

Drawing Pictures

```
void       DrawPicture(PicHandle myPicture,const Rect *dstRect)
PicHandle  GetPicture(Integer picID);
```

Collecting Picture Information

```
OSErr      GetPictInfo(PicHandle thePictHandle,PictInfo *thePictInfo,short verb,
           short colorsRequested,short colorPickMethod,short version);
OSErr      GetPixMapInfo(PixMapHandle thePixMapHandle,PictInfo *thePictInfo,short verb,
           short colorsRequested,short colorPickMethod,short version);
OSErr      NewPictInfo(PictInfoID *thePictInfoID,short verb,short colorsRequested,
           short colorPickMethod,short version);
OSErr      RecordPictInfo(PictInfoID thePictInfoID,PicHandle thePictHandle);
OSErr      RecordPixMapInfo(PictInfoID thePictInfoID,PixMapHandle thePixMapHandle);
OSErr      RetrievePictInfo(PictInfoID thePictInfoID,PictInfo *thePictInfo,
           short colorsRequested);
OSErr      DisposPictInfo(PictInfoID thePictInfoID);
```

Main Constants, Data Types and Functions-Cursors

Constants

```
iBeamCursor  = 1
crossCursor  = 2
plusCursor   = 3
watchCursor  = 4
```

Data Types

Cursor

```
struct Cursor
{
  Bits16       data;
  Bits16       mask;
  Point        hotSpot;
} ;
typedef struct Cursor Cursor;
typedef Cursor *CursPtr;
typedef CursPtr *CursHandle;
```

CCrsr

```
struct CCrsr
{
  short        crsrType;     // Type of cursor.
  PixMapHandle crsrMap;      // The cursor's pixmap.
  Handle       crsrData;     // Cursor's data.
  Handle       crsrXData;    // Expanded cursor data.
  short        crsrXValid;   // Depth of expanded data (0 if none).
  Handle       crsrXHandle;  // Future use.
  Bits16       crsr1Data;    // One-bit cursor.
  Bits16       crsrMask;     // Cursor's mask.
  Point        crsrHotSpot;  // Cursor's hotspot.
  long         crsrXTable;   // Private.
  long         crsrID;       // Private.
} ;
typedef struct CCrsr CCrsr;
typedef CCrsr *CCrsrPtr;
typedef CCrsrPtr *CCrsrHandle;
```

Acur

```
struct Acur
{
  short        n;            // Number of cursors (frames).
  short       index;         // (Reserved.)
  short       frame1;        // 'CURS' resource ID for frame #1.
  short       fill1;         // (Recerved.)
  short       frame2;        // 'CURS' resource ID for frame #2.
  short       fill2;         // (Reserved.)
  short       frameN;        // 'CURS' resource ID for frame #n.
  short       fillN;         // (Reserved.)
} ;
typedef struct Acur acur, *acurPtr, **acurHandle;
```

Functions

Initializing Cursors

```
void         InitCursor(void);
void         InitCursorCtl(acurHandle newCursors);
```

Changing Black-and-White Cursors

```
CursHandle   GetCursor(short cursorID);
void         SetCursor(const Cursor *crsr);
```

Changing Color Cursors

```
CCrsrHandle  GetCCursor(short crsrID);
void         SetCCursor(CCrsrHandle cCrsr);
void         AllocCursor(void)
void         DisposCCursor(CCrsrHandle cCrsr);
void         DisposeCCursor(CCrsrHandle cCrsr);
```

Hiding, Showing, and Animating Cursors

```
void         HideCursor(void);
void         ShowCursor(void);
void         ObscureCursor(void);
void         ShieldCursor(const Rect *shieldRect,Point offsetPt);
void         RotateCursor(long counter);
pascal       void SpinCursor(short increment);
```

Appearance Manager Constants, Data Types and Functions-Cursors

Constants

```
KThemeArrowCursor                 = 0
KThemeCopyArrowCursor             = 1
KThemeAliasArrowCursor            = 2
KThemeContextualMenuArrowCursor   = 3
KThemeIBeamCursor                 = 4
KThemeCrossCursor                 = 5
KThemePlusCursor                  = 6
KThemeWatchCursor                 = 7    Can animate
KThemeClosedHandCursor            = 8
KThemeOpenHandCursor              = 9
KThemePointingHandCursor          = 10
KThemeCountingUpHandCursor        = 11   Can animate
KThemeCountingDownHandCursor      = 12   Can animate
KThemeCountingUpAndDownHandCursor = 13   Can animate
KThemeSpinningCursor              = 14   Can Animate
KThemeResizeLeftCursor            = 15
KThemeResizeRightCursor           = 16
KThemeResizeLeftRightCursor       = 17
```

Data Types

```
typedef UInt32 ThemeCursor;
```

Functions

```
OSStatus  SetThemeCursor(ThemeCursor inCursor);
OSStatus  SetAnimatedThemeCursor(ThemeCursor inCursor,UInt32 inAnimationStep);
```

Mac OS X Only

```
void      QDDisplayWaitCursor(Boolean forceWaitCursor);
```

Main Constants, Data Types and Functions-Icons

Constants

Types for Icon Families

```
kLarge1BitMask              = FOUR_CHAR_CODE('ICN#')
kLarge4BitData              = FOUR_CHAR_CODE('icl4')
kLarge8BitData              = FOUR_CHAR_CODE('icl8')
kSmall1BitMask              = FOUR_CHAR_CODE('ics#')
kSmall4BitData              = FOUR_CHAR_CODE('ics4')
kSmall8BitData              = FOUR_CHAR_CODE('ics8')
kMini1BitMask               = FOUR_CHAR_CODE('icm#')
kMini4BitData               = FOUR_CHAR_CODE('icm4')
kMini8BitData               = FOUR_CHAR_CODE('icm8')
```

IconAlignmentType Values

```
kAlignNone                  = 0x00
kAlignVerticalCenter        = 0x01
kAlignTop                   = 0x02
kAlignBottom                = 0x03
kAlignHorizontalCenter      = 0x04
kAlignAbsoluteCenter        = kAlignVerticalCenter | kAlignHorizontalCenter
kAlignCenterTop             = kAlignTop            | kAlignHorizontalCenter
kAlignCenterBottom          = kAlignBottom         | kAlignHorizontalCenter
kAlignLeft                  = 0x08
kAlignCenterLeft            = kAlignVerticalCenter | kAlignLeft
kAlignTopLeft               = kAlignTop            | kAlignLeft
kAlignBottomLeft            = kAlignBottom         | kAlignLeft
kAlignRight                 = 0x0C
kAlignCenterRight           = kAlignVerticalCenter | kAlignRight
kAlignTopRight              = kAlignTop            | kAlignRight
kAlignBottomRight           = kAlignBottom         | kAlignRight
```

IconTransformType Values

```
kTransformNone              = 0x00
kTransformDisabled          = 0x01
kTransformOffline           = 0x02
kTransformOpen              = 0x03
kTransformLabel1            = 0x0100
kTransformLabel2            = 0x0200
kTransformLabel3            = 0x0300
kTransformLabel4            = 0x0400
kTransformLabel5            = 0x0500
kTransformLabel6            = 0x0600
kTransformLabel7            = 0x0700
kTransformSelected          = 0x4000
kTransformSelectedDisabled = kTransformSelected | kTransformDisabled
kTransformSelectedOffline  = kTransformSelected | kTransformOffline
kTransformSelectedOpen     = kTransformSelected | kTransformOpen
```

IconSelectorValue Masks

```
kSelectorLarge1Bit          = 0x00000001
kSelectorLarge4Bit          = 0x00000002
kSelectorLarge8Bit          = 0x00000004
kSelectorSmall1Bit          = 0x00000100
```

```
kSelectorSmall4Bit         = 0x00000200
kSelectorSmall8Bit         = 0x00000400
kSelectorMini1Bit          = 0x00010000
kSelectorMini4Bit          = 0x00020000
kSelectorMini8Bit          = 0x00040000
kSelectorAllLargeData      = 0x000000FF
kSelectorAllSmallData      = 0x0000FF00
kSelectorAllMiniData       = 0x00FF0000
kSelectorAll1BitData       = kSelectorLarge1Bit | kSelectorSmall1Bit | kSelectorMini1Bit
kSelectorAll4BitData       = kSelectorLarge4Bit | kSelectorSmall4Bit | kSelectorMini4Bit
kSelectorAll8BitData       = kSelectorLarge8Bit | kSelectorSmall8Bit | kSelectorMini8Bit
kSelectorAllAvailableData  = (long)0xFFFFFFFF
```

Data Types

```
typedef short   IconAlignmentType;
typedef short   IconTransformType;
typedef UInt32  IconSelectorValue;
typedef Handle  IconSuiteRef;
typedef Handle  IconCacheRef;
```

CIcon

```
struct CIcon
{
  PixMap iconPMap;         // Icon's pixMap.
  BitMap iconMask;         // Icon's mask.
  BitMap iconBMap;         // Icon's bitMap.
  Handle iconData;         // Icon's data.
  short  iconMaskData[1];  // Icon's mask and BitMap data.
} ;
typedef struct CIcon CIcon;
typedef CIcon *CIconPtr;
typedef CIconPtr *CIconHandle;
```

Functions

Drawing Icons from Resources

```
OSErr    PlotIconID(constRect *theRect,IconAlignmentType align,IconTransformType transform,
         short theResID);
void     PlotIcon(const Rect *theRect,Handle theIcon);
OSErr    PlotIconHandle(const Rect *theRect,IconAlignmentType align,
         IconTransformType transform,Handle theIcon);
void     PlotCIcon(const Rect *theRect,CIconHandle theIcon);
OSErr    PlotCIconHandle(const Rect *theRect,IconAlignmentType align,
         IconTransformType transform,CIconHandle theIcon);
OSErr    PlotSICNHandle(const Rect *theRect,IconAlignmentType align,
         IconTransformType transform,Handle theSICN);
```

Getting Icons from Resources Which Do Not Belong to an Icon Family

```
Handle       GetIcon(short iconID);
CIconHandle  GetCIcon(short iconID);
```

Disposing of Icons

```
OSErr    DisposeCIcon(CIconHandle theIcon);
```

Creating an Icon Suite

```
OSErr    GetIconSuite(Handle *theIconSuite,short theResID,IconSelectorValue selector);
```

```
OSErr    NewIconSuite(Handle *theIconSuite);
OSErr    AddIconToSuite(Handle theIconData,Handle theSuite,ResType theType);
```

Getting Icons from an Icon Suite

```
OSErr    GetIconFromSuite(Handle *theIconData,Handle theSuite,ResType theType);
```

Drawing Icons from an Icon Suite

```
OSErr    PlotIconSuite(const Rect *theRect,IconAlignmentType align,
         IconTransformType transform,Handle theIconSuite);
```

Performing Operations on Icons in an Icon Suite

```
OSErr    ForEachIconDo(handle theSuite,IconSelectorValue selector, IconActionUPP action,
         void *yourDataPtr);
```

Disposing of Icon Suites

```
OSErr    DisposeIconSuite(Handle theIconSuite,Boolean disposeData);
```

Converting an Icon Mask to a Region

```
OSErr    IconSuiteToRgn(RgnHandle theRgn,const Rect *iconRect,
         IconAlignmentType align,Handle theIconSuite);
OSErr    IconIDToRegion(RgnHandle theRgn,const Rect *iconRect,
         IconAlignmentType align,short iconID);
```

Determining Whether a Point or Rectangle Is Within an Icon

```
Boolean  PtInIconSuite(Point testPt,const Rect *iconRect,IconAlignmentType align,
         Handle theIconSuite);
Boolean  PtInIconID(Point testPt,const Rect *iconRect,IconAlignmentType align,
         short iconID);
Boolean  RectInIconSuite(const Rect *testRect,const Rect *iconRect,IconAlignmentType align,
         Handle theIconSuite);
Boolean  RectInIconID(const Rect *testRect,const Rect *iconRect,IconAlignmentType align,
         short iconID);
```

Working with Icon Caches

```
OSErr    MakeIconCache(Handle *theHandle,IconGetterProcPtr makeIcon,void *yourDataPtr);
OSErr    LoadIconCache(const Rect *theRect,IconAlignmentType align,
         IconTransformType transform,Handle theIconCache);
```

Demonstration Program GworldPicCursIcn Listing

```
//
******************************************************************************************
// GWorldPicCursIcon.c                                                    CLASSIC EVENT
MODEL
//
******************************************************************************************
//
// This program demonstrates offscreen graphics world, picture, cursor, cursor shape change,
// animated cursor, and icon operations as a result of the user choosing items from a
// Demonstration menu. It also demonstrates a modal dialog-based About... box containing a
// picture.
//
// To keep the non-demonstration code to a minimum, the program contains no functions for
// updating the window or for responding to activate and operating system events.
//
// The program utilizes the following resources:
//
// •  A 'plst' resource.
//
// •  An 'MBAR' resource and associated 'MENU' resources (preload, non-purgeable).
//
// •  A 'WIND' resource (purgeable) (initially visible).
//
// •  An 'acur' resource (purgeable).
//
// •  'CURS' resources associated with the 'acur' resource (preload, purgeable).
//
// •  Two 'cicn' resources (purgeable), one for the Icons menu item and one for drawing in
the
//    window.
//
// •  Two icon family resources (purgeable), both for drawing in the window.
//
// •  A 'DLOG' resource (purgeable) and an associated 'DITL' resource (purgeable) and 'PICT'
//    resource for an About GWorldPicCursIcon... dialog box.
//
// •  A 'STR#' resource (purgeable) containing transform constants.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
//
******************************************************************************************

//.....................................................................
includes

#include <Carbon.h>

// .....................................................................
defines

#define rMenubar                 128
#define rWindow                  128
#define mAppleApplication        128
#define  iAbout                  1
#define mFile                    129
#define  iQuit                   12
#define mDemonstration           131
#define  iOffScreenGWorld1       1
```

```
#define  iOffScreenGWorld2        2
#define  iPicture                 3
#define  iCursor                  4
#define  iAnimatedCursor1         5
#define  iAnimatedCursor2         6
#define  iAnimatedCursorOSX       7
#define  iIcon                    8
#define rBeachBallCursor          128
#define rPicture                  128
#define rTransformStrings         128
#define rIconFamily1              128
#define rIconFamily2              129
#define rColorIcon               128
#define rAboutDialog              128
#define kSleepTime                1
#define kBeachBallTickInterval    5
#define kCountingHandTickInterval 30
#define MAX_UINT32                0xFFFFFFFF
#define topLeft(r)                (((Point *) &(r))[0])
#define botRight(r)               (((Point *) &(r))[1])
// ..........................................................................
typedefs

typedef struct
{
  SInt16     numberOfFrames;
  SInt16     whichFrame;
  CursHandle frame[];
}  animCurs, *animCursPtr, **animCursHandle;

// ..................................................................... global
variables

Boolean        gRunningOnX = false;
WindowRef      gWindowRef;
Boolean        gDone;
SInt32         gSleepTime;
RgnHandle      gCursorRegion;
Boolean        gCursorRegionsActive     = false;
Boolean        gAnimatedCursor1Active   = false;
Boolean        gAnimatedCursor2Active   = false;
Boolean        gAnimatedCursorOSXActive = false;
animCursHandle gAnimCursHdl;
SInt16         gAnimCursTickInterval;
SInt32         gAnimCursLastTick;
RGBColor       gBlackColor = {  0x0000, 0x0000, 0x0000 } ;
RGBColor       gWhiteColor = {  0xFFFF, 0xFFFF, 0xFFFF } ;
RGBColor       gBeigeColor = {  0xF000, 0xE300, 0xC200 } ;
RGBColor       gBlueColor  = {  0x4444, 0x4444, 0x9999 } ;

// ..................................................................... function
prototypes

void    main                  (void);
void    doPreliminaries       (void);
OSErr   quitAppEventHandler   (AppleEvent *,AppleEvent *,SInt32);
void    eventLoop             (void);
void    doIdle                (void);
void    doEvents              (EventRecord *);
void    doMenuChoice          (SInt32);
void    doOffScreenGWorld1    (void);
void    doOffScreenGWorld2    (void);
void    doPicture             (void);
void    doCursor              (void);
void    doChangeCursor        (WindowRef,RgnHandle);
void    doAnimatedCursor1     (void);
```

```
void    doAnimatedCursor2     (void);
Boolean doGetAnimCursor       (SInt16,SInt16);
void    doIncrementAnimCursor (void);
void    doReleaseAnimCursor   (void);
void    doAnimatedCursorOSX   (void);
void    doIcon                (void);
void    doAboutDialog         (void);
void    doDrawStuff           (void);
UInt16  doRandomNumber        (UInt16,UInt16);
// ****************************************************************************
main

void  main(void)
{
  UInt32        seconds;
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;

  // ..................................................................... initialize
managers

  doPreliminaries();

  // ............................................................... seed random number
generator

  GetDateTime(&seconds);
  SetQDGlobalsRandomSeed(seconds);

  // ................................................................ set up menu bar
and menus

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    ExitToShell();
  SetMenuBar(menubarHdl);
  DrawMenuBar();

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
      DisableMenuItem(menuRef,0);
    }

    menuRef = GetMenuRef(mDemonstration);
    if(menuRef != NULL)
      EnableMenuItem(menuRef,iAnimatedCursorOSX);

    gRunningOnX = true;
  }

  // ..........................................................................
open window

  if(!(gWindowRef = GetNewCWindow(rWindow,NULL,(WindowRef)-1)))
    ExitToShell();

  SetPortWindowPort(gWindowRef);
  TextSize(10);
  // ..................................................................... enter event
```

```
loop

  eventLoop();
}

// ********************************************************************** do
preliminaries

void  doPreliminaries(void)
{
  OSErr osError;

  MoreMasterPointers(64);
  InitCursor();
  FlushEvents(everyEvent,0);

  osError = AEInstallEventHandler(kCoreEventClass,kAEQuitApplication,
                           NewAEEventHandlerUPP((AEEventHandlerProcPtr)
quitAppEventHandler),
                           0L,false);
  if(osError != noErr)
    ExitToShell();
}

// **********************************************************************
doQuitAppEvent

OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  OSErr    osError;
  DescType returnedType;
  Size     actualSize;

  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,
NULL,0,
                              &actualSize);

  if(osError == errAEDescNotFound)
  {
    gDone = true;
    osError = noErr;
  }
  else if(osError == noErr)
    osError = errAEParamMissed;

  return osError;
}

// ****************************************************************************
eventLoop

void  eventLoop(void)
{
  EventRecord eventStructure;
  Boolean     gotEvent;

  gDone = false;
  gSleepTime = MAX_UINT32;
  gCursorRegion = NULL;
  while(!gDone)
  {
    gotEvent = WaitNextEvent(everyEvent,&eventStructure,gSleepTime,gCursorRegion);
    if(gotEvent)
      doEvents(&eventStructure);
    else
    {
```

```
      if(eventStructure.what == nullEvent)
        doIdle();
    }
  }
}

// ********************************************************************************
doIdle

void  doIdle(void)
{
  if(gAnimatedCursor1Active || gAnimatedCursor2Active)
    doIncrementAnimCursor();
}

// ********************************************************************************
doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
  WindowRef      windowRef;
  WindowPartCode partCode;

  switch(eventStrucPtr->what)
  {
    case kHighLevelEvent:
      AEProcessAppleEvent(eventStrucPtr);
      break;

    case mouseDown:
      partCode = FindWindow(eventStrucPtr->where,&windowRef);
      switch(partCode)
      {
        case inMenuBar:
          doMenuChoice(MenuSelect(eventStrucPtr->where));
          break;

        case inContent:
          if(windowRef != FrontWindow())
            SelectWindow(windowRef);
          break;

        case inDrag:
          DragWindow(windowRef,eventStrucPtr->where,NULL);
          if(gCursorRegionsActive)
            doChangeCursor(windowRef,gCursorRegion);
          break;
      }
      break;
    case keyDown:
      if((eventStrucPtr->modifiers & cmdKey) != 0)
        doMenuChoice(MenuEvent(eventStrucPtr));
      break;

    case updateEvt:
      BeginUpdate((WindowRef) eventStrucPtr->message);
      EndUpdate((WindowRef) eventStrucPtr->message);
      break;

    case osEvt:
      switch((eventStrucPtr->message >> 24) & 0x000000FF)
      {
        case suspendResumeMessage:
          if((eventStrucPtr->message & resumeFlag) == 1)
            SetThemeCursor(kThemeArrowCursor);
          break;
```

```
        case mouseMovedMessage:
          if(gCursorRegionsActive)
            doChangeCursor(FrontWindow(),gCursorRegion);
          break;
      }
      break;
  }
}

// ****************************************************************************
doMenuChoice

void  doMenuChoice(SInt32 menuChoice)
{
  MenuID        menuID;
  MenuItemIndex menuItem;

  menuID = HiWord(menuChoice);
  menuItem = LoWord(menuChoice);

  if(menuID == 0)
    return;

  if(gAnimatedCursor1Active || gAnimatedCursor2Active)
  {
    if(gAnimatedCursor2Active)
      doReleaseAnimCursor();

    SetThemeCursor(kThemeArrowCursor);
    gSleepTime = MAX_UINT32;

    gAnimatedCursor1Active = false;
    gAnimatedCursor2Active = false;
  }

  if(gAnimatedCursorOSXActive)
    doAnimatedCursorOSX();
  if(gCursorRegionsActive == true)
  {
    gCursorRegionsActive = false;
    DisposeRgn(gCursorRegion);
    gCursorRegion = NULL;
  }

  switch(menuID)
  {
    case mAppleApplication:
      if(menuItem == iAbout)
        doAboutDialog();
      break;

    case mFile:
      if(menuItem == iQuit)
        gDone = true;
      break;

    case mDemonstration:
      switch(menuItem)
      {
        case iOffScreenGWorld1:
          doOffScreenGWorld1();
          break;

        case iOffScreenGWorld2:
          doOffScreenGWorld2();
```

```
          break;

        case iPicture:
          doPicture();
          break;

        case iCursor:
          doCursor();
          break;

        case iAnimatedCursor1:
          doAnimatedCursor1();
          break;

        case iAnimatedCursor2:
          doAnimatedCursor2();
          break;

        case iAnimatedCursorOSX:
          doAnimatedCursorOSX();
          break;

        case iIcon:
          doIcon();
          break;
      }
      break;
  }
  HiliteMenu(0);
}

// **************************************************************************
doOffScreenGWorld1

void  doOffScreenGWorld1(void)
{
  Rect         portRect, sourceRect, destRect;
  GrafPtr      windowPortPtr;
  GDHandle     deviceHdl;
  QDErr        qdErr;
  GWorldPtr    gworldPortPtr;
  PixMapHandle gworldPixMapHdl, windowPixMapHdl;
  Boolean      lockPixResult;

  // ..........................................................................
draw in window

  SetWTitle(gWindowRef,"\pTime-consuming drawing operation");

  if(!gRunningOnX)
    SetThemeCursor(kThemeWatchCursor);

  doDrawStuff();

  if(!gRunningOnX)
    SetThemeCursor(kThemeArrowCursor);

  SetWTitle(gWindowRef,"\pClick mouse to repeat in offscreen graphics port");
  QDFlushPortBuffer(GetWindowPort(gWindowRef),NULL);

  while(!Button()) ;

  if(!gRunningOnX)
    SetThemeCursor(kThemeWatchCursor);

  GetWindowPortBounds(gWindowRef,&portRect);
```

```
  RGBBackColor(&gBlueColor);
  EraseRect(&portRect);
  RGBForeColor(&gWhiteColor);
  MoveTo(190,180);
  DrawString("\pPlease Wait. Drawing in offscreen graphics port.");

  // ..................................... draw in offscreen graphics port and copy to
window

  // ......................... save current graphics world and create offscreen graphics
world

  GetGWorld(&windowPortPtr,&deviceHdl);

  qdErr = NewGWorld(&gworldPortPtr,0,&portRect,NULL,NULL,0);
  if(gworldPortPtr == NULL || qdErr != noErr)
  {
    SysBeep(10);
    return;
  }
  SetGWorld(gworldPortPtr,NULL);

  // ................... lock pixel image for duration of drawing and erase offscreen to
white

  gworldPixMapHdl = GetGWorldPixMap(gworldPortPtr);
  if(!(lockPixResult = LockPixels(gworldPixMapHdl)))
  {
    SysBeep(10);
    return;
  }

  EraseRect(&portRect);

  // ................................................ draw into the offscreen graphics
port

  doDrawStuff();

  // ........................................................ restore saved graphics
world

  SetGWorld(windowPortPtr,deviceHdl);

  // ................................................ set source and destination
rectangles

  GetPortBounds(gworldPortPtr,&sourceRect);
  GetPortBounds(windowPortPtr,&destRect);

  // ........................................................ get window port's pixel
map

  windowPixMapHdl = GetGWorldPixMap(windowPortPtr);

  // .............. ensure background color is white and foreground color in black, then copy

  RGBBackColor(&gWhiteColor);
  RGBForeColor(&gBlackColor);

  CopyBits((BitMap *) *gworldPixMapHdl,
           (BitMap *) *windowPixMapHdl,
           &sourceRect,&destRect,srcCopy,NULL);

  if(QDError() != noErr)
```

```
    SysBeep(10);

  // ..................................................... clean up

  UnlockPixels(gworldPixMapHdl);
  DisposeGWorld(gworldPortPtr);

  if(!gRunningOnX)
    SetThemeCursor(kThemeArrowCursor);

  SetWTitle(gWindowRef,"\pOffscreen Graphics Worlds, Pictures,  Cursors and Icons");
  QDFlushPortBuffer(GetWindowPort(gWindowRef),NULL);
}
// ************************************************************************
doOffScreenGWorld2

void  doOffScreenGWorld2(void)
{
  PicHandle    picture1Hdl,picture2Hdl;
  Rect         portRect;
  Rect         sourceRect, maskRect, maskDisplayRect, dest1Rect, dest2Rect, destRect;
  GrafPtr      windowPortPtr;
  GDHandle     deviceHdl;
  QDErr        qdErr;
  GWorldPtr    gworldPortPtr;
  PixMapHandle gworldPixMapHdl, windowPixMapHdl;
  RgnHandle    region1Hdl, region2Hdl, regionHdl;
  SInt16       a, sourceMode;

  RGBBackColor(&gBeigeColor);
  GetWindowPortBounds(gWindowRef,&portRect);
  EraseRect(&portRect);

  // ..................................... get the source picture and draw it in the
window

  if(!(picture1Hdl = GetPicture(rPicture)))
    ExitToShell();
  HNoPurge((Handle) picture1Hdl);
  SetRect(&sourceRect,116,35,273,147);
  DrawPicture(picture1Hdl,&sourceRect);
  HPurge((Handle) picture1Hdl);
  MoveTo(116,32);
  DrawString("\pSource image");

  // ............................... save current graphics world and create offscreen
graphics world

  GetGWorld(&windowPortPtr,&deviceHdl);

  SetRect(&maskRect,0,0,157,112);

  qdErr = NewGWorld(&gworldPortPtr,0,&maskRect,NULL,NULL,0);
  if(gworldPortPtr == NULL || qdErr != noErr)
  {
    SysBeep(10);
    return;
  }

  SetGWorld(gworldPortPtr,NULL);

  // ......................... lock pixel image for duration of drawing and erase
offscreen to white

  gworldPixMapHdl = GetGWorldPixMap(gworldPortPtr);
```

```
  if(!(LockPixels(gworldPixMapHdl)))
  {
    SysBeep(10);
    return;
  }
  GetPortBounds(gworldPortPtr,&portRect);
  EraseRect(&portRect);

  // ..................................... get mask picture and draw it in offscreen
graphics port

  if(!(picture2Hdl = GetPicture(rPicture + 1)))
    ExitToShell();
  HNoPurge((Handle) picture2Hdl);
  DrawPicture(picture2Hdl,&maskRect);

  // ............................................................ also draw it in the
window

  SetGWorld(windowPortPtr,deviceHdl);
  SetRect(&maskDisplayRect,329,35,485,146);
  DrawPicture(picture2Hdl,&maskDisplayRect);
  HPurge((Handle) picture2Hdl);
  MoveTo(329,32);
  DrawString("\pCopy of offscreen mask");

  // ................................... define an oval-shaped region and a round
rectangle-shaped region

  SetRect(&dest1Rect,22,171,296,366);
  region1Hdl = NewRgn();
  OpenRgn();
  FrameOval(&dest1Rect);
  CloseRgn(region1Hdl);

  SetRect(&dest2Rect,308,171,582,366);
  region2Hdl = NewRgn();
  OpenRgn();
  FrameRoundRect(&dest2Rect,100,100);
  CloseRgn(region2Hdl);

  SetWTitle(GetWindowFromPort(windowPortPtr),"\pClick mouse to copy");
  QDFlushPortBuffer(GetWindowPort(gWindowRef),NULL);
  while(!Button()) ;

  // ..................................................... get window port's pixel
map

  windowPixMapHdl = GetGWorldPixMap(windowPortPtr);

  // ......... set background and foreground color, then copy source to destination using
mask

  RGBForeColor(&gBlackColor);
  RGBBackColor(&gWhiteColor);

  for(a=0;a<2;a++)
  {
    if(a == 0)
    {
      regionHdl = region1Hdl;
      destRect = dest1Rect;
      sourceMode = srcCopy;
      MoveTo(22,168);
      DrawString("\pBoolean source mode srcCopy");
    }
```

```
    else
    {
      regionHdl = region2Hdl;
      destRect = dest2Rect;
      sourceMode = srcXor;
      MoveTo(308,168);
      DrawString("\pBoolean source mode srcXor");
    }

    CopyDeepMask((BitMap *) *windowPixMapHdl,
                 (BitMap *) *gworldPixMapHdl,
                 (BitMap *) *windowPixMapHdl,
                 &sourceRect,&maskRect,&destRect,sourceMode + ditherCopy,regionHdl);

    if(QDError() != noErr)
      SysBeep(10);
  }

  //.....................................................................................
clean up

  UnlockPixels(gworldPixMapHdl);
  DisposeGWorld(gworldPortPtr);

  ReleaseResource((Handle) picture1Hdl);
  ReleaseResource((Handle) picture2Hdl);
  DisposeRgn(region1Hdl);
  DisposeRgn(region2Hdl);

  SetWTitle(gWindowRef,"\pOffscreen Graphics Worlds, Pictures,  Cursors and Icons");
  QDFlushPortBuffer(GetWindowPort(gWindowRef),NULL);
}

// ***************************************************************************
doPicture

void  doPicture(void)
{
  Rect           portRect, pictureRect, theRect;
  OpenCPicParams picParams;
  RgnHandle      oldClipRgn;
  PicHandle      pictureHdl;
  SInt16         a, left, top, right, bottom, random;
  RGBColor       theColor;
  PictInfo       pictInfo;
  Str255         theString;

  RGBBackColor(&gWhiteColor);
  GetWindowPortBounds(gWindowRef,&portRect);
  EraseRect(&portRect);
  // ..................................................................... define picture
rectangle

  pictureRect = portRect;
  pictureRect.right = (portRect.right - portRect.left) / 2;
  InsetRect(&pictureRect,10,10);

  // ..................................................................... set
clipping region

  oldClipRgn = NewRgn();
  GetClip(oldClipRgn);
  ClipRect(&pictureRect);

  // ................................................ set up OpenCPicParams structure
```

```
  picParams.srcRect  = pictureRect;
  picParams.hRes    = 0x00480000;
  picParams.vRes    = 0x00480000;
  picParams.version = -2;

  // ..................................................................... record picture

  pictureHdl = OpenCPicture(&picParams);

  RGBBackColor(&gBlueColor);
  EraseRect(&pictureRect);

  for(a=0;a<300;a++)
  {
    theRect = pictureRect;

    theColor.red   = doRandomNumber(0,65535);
    theColor.green = doRandomNumber(0,65535);
    theColor.blue  = doRandomNumber(0,65535);
    RGBForeColor(&theColor);

    left = doRandomNumber(10,theRect.right);
    top = doRandomNumber(10,theRect.bottom);
    right = doRandomNumber(left,theRect.right);
    bottom = doRandomNumber(top,theRect.bottom);
    SetRect(&theRect,left,top,right,bottom);

    PenMode(doRandomNumber(addOver,adMin));

    random = doRandomNumber(0,5);

    if(random == 0)
    {
      MoveTo(left,top);
      LineTo(right - 1,bottom - 1);
    }
    else if(random == 1)
      PaintRect(&theRect);
    else if(random == 2)
      PaintRoundRect(&theRect,30,30);
    else if(random == 3)
      PaintOval(&theRect);
    else if(random == 4)
      PaintArc(&theRect,0,300);
    else if(random == 5)
    {
      TextSize(doRandomNumber(10,70));
      MoveTo(left,right);
      DrawString("\pPICTURE");
    }
  }

  // ..................................... stop recording, draw picture, restore saved
clipping region

  ClosePicture();

  DrawPicture(pictureHdl,&pictureRect);

  SetClip(oldClipRgn);
  DisposeRgn(oldClipRgn);

  // ........................................... display some information from the
PictInfo structure

  RGBForeColor(&gBlueColor);
```

```
RGBBackColor(&gBeigeColor);
PenMode(patCopy);
OffsetRect(&pictureRect,300,0);
EraseRect(&pictureRect);
FrameRect(&pictureRect);
TextSize(10);

if(GetPictInfo(pictureHdl,&pictInfo,recordFontInfo + returnColorTable,1,systemMethod,0))
  SysBeep(10);

MoveTo(380,70);
DrawString("\pSome Picture Information:");

MoveTo(380,100);
DrawString("\pLines: ");
NumToString(pictInfo.lineCount,theString);
DrawString(theString);

MoveTo(380,115);
DrawString("\pRectangles: ");
NumToString((long) pictInfo.rectCount,theString);
DrawString(theString);

MoveTo(380,130);
DrawString("\pRound rectangles: ");
NumToString(pictInfo.rRectCount,theString);
DrawString(theString);

MoveTo(380,145);
DrawString("\pOvals: ");
NumToString(pictInfo.ovalCount,theString);
DrawString(theString);

MoveTo(380,160);
DrawString("\pArcs: ");
NumToString(pictInfo.arcCount,theString);
DrawString(theString);

MoveTo(380,175);
DrawString("\pPolygons: ");
NumToString(pictInfo.polyCount,theString);
DrawString(theString);

MoveTo(380,190);
DrawString("\pRegions: ");
NumToString(pictInfo.regionCount,theString);
DrawString(theString);

MoveTo(380,205);
DrawString("\pText strings: ");
NumToString(pictInfo.textCount,theString);
DrawString(theString);

MoveTo(380,220);
DrawString("\pUnique fonts: ");
NumToString(pictInfo.uniqueFonts,theString);
DrawString(theString);

MoveTo(380,235);
DrawString("\pUnique colors: ");
NumToString(pictInfo.uniqueColors,theString);
DrawString(theString);

MoveTo(380,250);
DrawString("\pFrame rectangle left: ");
NumToString(pictInfo.sourceRect.left,theString);
```

```
  DrawString(theString);

  MoveTo(380,265);
  DrawString("\pFrame rectangle top: ");
  NumToString(pictInfo.sourceRect.top,theString);
  DrawString(theString);

  MoveTo(380,280);
  DrawString("\pFrame rectangle right: ");
  NumToString(pictInfo.sourceRect.right,theString);
  DrawString(theString);

  MoveTo(380,295);
  DrawString("\pFrame rectangle bottom: ");
  NumToString(pictInfo.sourceRect.bottom,theString);
  DrawString(theString);
  QDFlushPortBuffer(GetWindowPort(gWindowRef),NULL);

  // ..... release memory occupied by Picture
structure

  KillPicture(pictureHdl);
}

// ****************************************************************************
doCursor

void  doCursor(void)
{
  Rect   portRect, cursorRect;
  SInt16 a;

  RGBBackColor(&gBlueColor);
  GetWindowPortBounds(gWindowRef,&portRect);
  EraseRect(&portRect);

  cursorRect = portRect;

  for(a=0;a<3;a++)
  {
    InsetRect(&cursorRect,40,40);

    if(a == 0 || a == 2)
      RGBBackColor(&gBeigeColor);
    else
      RGBBackColor(&gBlueColor);

    EraseRect(&cursorRect);
  }

  RGBForeColor(&gBeigeColor);
  MoveTo(10,20);
  DrawString("\pArrow cursor region");
  RGBForeColor(&gBlueColor);
  MoveTo(50,60);
  DrawString("\pIBeam cursor region");
  RGBForeColor(&gBeigeColor);
  MoveTo(90,100);
  DrawString("\pCross cursor region");
  RGBForeColor(&gBlueColor);
  MoveTo(130,140);
  DrawString("\pPlus cursor region");

  gCursorRegion = NewRgn();
  doChangeCursor(gWindowRef,gCursorRegion);
```

```
  gCursorRegionsActive = true;
}
// ************************************************************************
doChangeCursor

void  doChangeCursor(WindowRef windowRef,RgnHandle cursorRegion)
{
  RgnHandle arrowCursorRgn;
  RgnHandle ibeamCursorRgn;
  RgnHandle crossCursorRgn;
  RgnHandle plusCursorRgn;
  Rect      cursorRect;
  GrafPtr   oldPort;
  Point     mousePosition;

  arrowCursorRgn = NewRgn();
  ibeamCursorRgn = NewRgn();
  crossCursorRgn = NewRgn();
  plusCursorRgn   = NewRgn();

  SetRectRgn(arrowCursorRgn,-32768,-32768,32766,32766);

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

  GetWindowPortBounds(windowRef,&cursorRect);
  LocalToGlobal(&topLeft(cursorRect));
  LocalToGlobal(&botRight(cursorRect));

  InsetRect(&cursorRect,40,40);
  RectRgn(ibeamCursorRgn,&cursorRect);
  DiffRgn(arrowCursorRgn,ibeamCursorRgn,arrowCursorRgn);

  InsetRect(&cursorRect,40,40);
  RectRgn(crossCursorRgn,&cursorRect);
  DiffRgn(ibeamCursorRgn,crossCursorRgn,ibeamCursorRgn);

  InsetRect(&cursorRect,40,40);
  RectRgn(plusCursorRgn,&cursorRect);
  DiffRgn(crossCursorRgn,plusCursorRgn,crossCursorRgn);

  GetGlobalMouse(&mousePosition);

  if(PtInRgn(mousePosition,ibeamCursorRgn))
  {
    SetThemeCursor(kThemeIBeamCursor);
    CopyRgn(ibeamCursorRgn,cursorRegion);
  }
  else if(PtInRgn(mousePosition,crossCursorRgn))
  {
    SetThemeCursor(kThemeCrossCursor);
    CopyRgn(crossCursorRgn,cursorRegion);
  }
  else if(PtInRgn(mousePosition,plusCursorRgn))
  {
    SetThemeCursor(kThemePlusCursor);
    CopyRgn(plusCursorRgn,cursorRegion);
  }
  else
  {
    SetThemeCursor(kThemeArrowCursor);
    CopyRgn(arrowCursorRgn,cursorRegion);
  }

  DisposeRgn(arrowCursorRgn);
  DisposeRgn(ibeamCursorRgn);
```

```
  DisposeRgn(crossCursorRgn);
  DisposeRgn(plusCursorRgn);

  SetPort(oldPort);
}

// ************************************************************************
doAnimatedCursor1

void  doAnimatedCursor1(void)
{
  Rect    portRect;
  Pattern whitePattern;

  BackColor(whiteColor);
  GetWindowPortBounds(gWindowRef,&portRect);
  FillRect(&portRect,GetQDGlobalsWhite(&whitePattern));

  gAnimCursTickInterval = kCountingHandTickInterval;
  gSleepTime = gAnimCursTickInterval;
  gAnimatedCursor1Active = true;

}

// ************************************************************************
doAnimatedCursor2

void  doAnimatedCursor2(void)
{
  Rect    portRect;
  Pattern whitePattern;
  SInt16  animCursResourceID, animCursTickInterval;

  BackColor(whiteColor);
  GetWindowPortBounds(gWindowRef,&portRect);
  FillRect(&portRect,GetQDGlobalsWhite(&whitePattern));

  animCursResourceID   = rBeachBallCursor;
  animCursTickInterval = kBeachBallTickInterval;

  if(doGetAnimCursor(animCursResourceID,animCursTickInterval))
  {
    gSleepTime = animCursTickInterval;
    gAnimatedCursor2Active = true;
  }
  else
    SysBeep(10);
}
// ************************************************************************
doGetAnimCursor

Boolean  doGetAnimCursor(SInt16 resourceID,SInt16 tickInterval)
{
  SInt16  cursorID, a = 0;
  Boolean noError = false;

  if((gAnimCursHdl = (animCursHandle) GetResource('acur',resourceID)))
  {
    noError = true;
    while((a < (*gAnimCursHdl)->numberOfFrames)  && noError)
    {
      cursorID = (SInt16) HiWord((SInt32) (*gAnimCursHdl)->frame[a]);
      (*gAnimCursHdl)->frame[a] = GetCursor(cursorID);
      if((*gAnimCursHdl)->frame[a])
        a++;
      else
```

```
        noError = false;
    }
  }

  if(noError)
  {
    gAnimCursTickInterval = tickInterval;
    gAnimCursLastTick = TickCount();
    (*gAnimCursHdl)->whichFrame = 0;
  }

  return noError;
}

// **********************************************************************
doIncrementAnimCursor

void  doIncrementAnimCursor(void)
{
  SInt32        newTick;
  static UInt32 animationStep;

  newTick = TickCount();
  if(newTick < (gAnimCursLastTick + gAnimCursTickInterval))
    return;

  if(gAnimatedCursor1Active)
  {
    SetAnimatedThemeCursor(kThemeCountingUpAndDownHandCursor,animationStep);
      animationStep++;
  }
  else if(gAnimatedCursor2Active)
  {
    SetCursor(*((*gAnimCursHdl)->frame[(*gAnimCursHdl)->whichFrame++]));
    if((*gAnimCursHdl)->whichFrame == (*gAnimCursHdl)->numberOfFrames)
      (*gAnimCursHdl)->whichFrame = 0;
  }
  gAnimCursLastTick = newTick;
}

// **********************************************************************
doReleaseAnimCursor

void  doReleaseAnimCursor(void)
{
  SInt16 a;

  for(a=0;a<(*gAnimCursHdl)->numberOfFrames;a++)
    ReleaseResource((Handle) (*gAnimCursHdl)->frame[a]);

  ReleaseResource((Handle) gAnimCursHdl);
}

// **********************************************************************
doAnimatedCursorOSX

void  doAnimatedCursorOSX(void)
{
  if(!gAnimatedCursorOSXActive)
  {
    QDDisplayWaitCursor(true);
    gAnimatedCursorOSXActive = true;
  }
  else
  {
    QDDisplayWaitCursor(false);
```

```
    gAnimatedCursorOSXActive = false;
  }
}

// ****************************************************************************************
doIcon

void  doIcon(void)
{
  Rect              portRect, theRect;
  SInt16            a, b, stringIndex = 1;
  IconTransformType transform = 0;
  Str255            theString;
  Handle            iconSuiteHdl;
  CIconHandle       ciconHdl;

  RGBForeColor(&gBlueColor);
  RGBBackColor(&gBeigeColor);
  GetWindowPortBounds(gWindowRef,&portRect);
  EraseRect(&portRect);

  // ..................................................................... PlotIconID with
transforms

  MoveTo(50,28);
  DrawString("\pPlotIconID with transforms");
  for(a=50;a<471;a+=140)
  {
    if(a == 190)
      transform = 16384;
    if(a == 330)
      transform = 256;

    for(b=0;b<4;b++)
    {
      if(a == 470 && b == 3)
        continue;

      GetIndString(theString,rTransformStrings,stringIndex++);
      MoveTo(a,b * 60 + 47);
      DrawString(theString);

      SetRect(&theRect,a,b * 60 + 50,a + 32,b * 60 + 82);
      PlotIconID(&theRect,0,transform,rIconFamily1);
      SetRect(&theRect,a + 40,b * 60 + 50,a + 56,b * 60 + 66);
      PlotIconID(&theRect,0,transform,rIconFamily1);
      SetRect(&theRect,a + 64,b * 60 + 50,a + 80,b * 60 + 62);
      PlotIconID(&theRect,0,transform,rIconFamily1);

      if(a >= 330)
        transform += 256;
      else
        transform ++;
    }
  }

  // ................................................... GetIconSuite and PlotIconSuite

  MoveTo(50,275);
  LineTo(550,275);
  MoveTo(50,299);
  DrawString("\pGetIconSuite and PlotIconSuite");

  GetIconSuite(&iconSuiteHdl,rIconFamily2,kSelectorAllLargeData);

  SetRect(&theRect,50,324,82,356);
```

```
  PlotIconSuite(&theRect,kAlignNone,kTransformNone,iconSuiteHdl);
  SetRect(&theRect,118,316,166,364);
  PlotIconSuite(&theRect,kAlignNone,kTransformNone,iconSuiteHdl);
  SetRect(&theRect,202,308,266,372);
  PlotIconSuite(&theRect,kAlignNone,kTransformNone,iconSuiteHdl);

  // ..................................................................... GetCIcon
and PlotCIcon

  MoveTo(330,299);
  DrawString("\pGetCIcon and PlotCIcon");

  ciconHdl = GetCIcon(rColorIcon);
  SetRect(&theRect,330,324,362,356);
  PlotCIcon(&theRect,ciconHdl);
  SetRect(&theRect,398,316,446,364);
  PlotCIcon(&theRect,ciconHdl);
  SetRect(&theRect,482,308,546,372);
  PlotCIcon(&theRect,ciconHdl);
}

// ****************************************************************************
doAboutDialog

void  doAboutDialog(void)
{
  DialogPtr dialogPtr;
  SInt16    itemHit;

  dialogPtr = GetNewDialog(rAboutDialog,NULL,(WindowRef)-1);
  ModalDialog(NULL,&itemHit);
  DisposeDialog(dialogPtr);
}

// ****************************************************************************
doDrawStuff

void  doDrawStuff(void)
{
  Rect     portRect, theRect;
  RGBColor theColor;
  SInt16   a, left, top, right, bottom, random;

  RGBBackColor(&gBlueColor);
  GetWindowPortBounds(gWindowRef,&portRect);
  EraseRect(&portRect);

  for(a=0;a<900;a++)
  {
    theRect = portRect;

    theColor.red   = doRandomNumber(0,65535);
    theColor.green = doRandomNumber(0,65535);
    theColor.blue  = doRandomNumber(0,65535);
    RGBForeColor(&theColor);

    left = doRandomNumber(0,theRect.right);
    top = doRandomNumber(0,theRect.bottom);
    right = doRandomNumber(left,theRect.right);
    bottom = doRandomNumber(top,theRect.bottom);
    SetRect(&theRect,left,top,right,bottom);

    PenMode(doRandomNumber(addOver,adMin));

    random = doRandomNumber(0,3);
```

```
    if(random == 0)
      PaintRect(&theRect);
    else if(random == 1)
      PaintRoundRect(&theRect,doRandomNumber(10,100),doRandomNumber(10,100));
    else if(random == 2)
      PaintOval(&theRect);
    else if(random == 3)
      PaintArc(&theRect,0,doRandomNumber(5,330));

    QDFlushPortBuffer(GetWindowPort(gWindowRef),NULL);
  }
}

// ********************************************************************************
doRandomNumber

UInt16  doRandomNumber(UInt16 minimum, UInt16 maximum)
{
  UInt16 randomNumber;
  SInt32 range, t;

  randomNumber = Random();
  range = maximum - minimum + 1;
  t = (randomNumber * range) / 65536;
  return (t + minimum);
}

//
********************************************************************************************
```

Demonstration Program GWorldPicCursIcn Comments

When this program is run, the user should

- Invoke the demonstrations by choosing items from the Demonstration menu, clicking the mouse when instructed to do so by the text in the window's title bar.
- Click outside and inside the window when the animated cursor demonstrations have been invoked.
- Choose the About… item in the Apple menu to display the About… dialog.
- Note that the Icons item in the Demonstration menu contains an icon.

On Mac OS 8/9, if the first offscreen graphics world demonstration does not work when the monitor color depth is set to Millions, increase the Minimum Heap Size set in the CodeWarrior project.

defines

Constants are established for the resource IDs of `'acur'`, `'PICT'`, `'STR#'`, icon family, and `'cicn'` resources, and a `'DLOG'` resource.

`kSleeptime` and `MAX_UINT32` will be assigned to `WaitNextEvent`'s sleep parameter at various points in the program. `kBeachBallTickInterval` represents the interval between frame changes for the first of three animated cursors. `kCountingHandTickInterval` represents the interval between frame changes for the second animated cursor.

typedefs

The data type `anumCurs` is identical to the structure of an `'acur'` resource.

Global Variables

In this program, the sleep and cursor region parameters in the `WaitNextEvent` call will be changed during program execution, hence the global variables `gSleepTime` and `gCursorRegion`. `gCursorRegion` will be assigned a reference to a region which will be passed in the `mouseRgn` parameter of the `WaitNextEvent` call. This relates to the cursor shape changing demonstration.

gAnimCursHdl will be assigned a handle to the animCurs structure used during the first animated cursor demonstration. gAnimCursTickInterval and gAnimCursLastTick also relate to the animated cursor demonstration.

main

Random numbers will be used in the function doPicture. The call to SetQDGlobalsRandomSeed seeds the random number generator with the value returned by the call to GetDateTime.

Note that error handling here and in other areas of the program is somewhat rudimentary: The program simply terminates, sometimes with a call to SysBeep().

eventLoop

Before the event loop is entered, gSleepTime is set to MAX_UINT32. Initially, therefore, the sleep parameter in the WaitNextEvent call is set to the maximum possible UInt32 value.

The global variable passed in the mouseRgn parameter of the WaitNextEvent call is assigned NULL so as to defeat the generation of mouse-moved events.

When WaitNextEvent returns 0 with a null event, the function doIdle is called.

doIdle

doIdle is called from the main event loop when WaitNextEvent returns 0 with a null event. If the active demonstration is one of the first two animated cursor demonstrations, the function doIncrementAnimCursor is called.

doEvents

In the inDrag case, after the call to DragWindow, and provided the cursor shape changing demonstration is currently under way, the function doChangeCursor is called.

The regions controlling the generation of mouse-moved events are defined in global coordinates, and are based on the window's port rectangle. Accordingly, when the window is moved, the new location of the port rectangle, in global coordinates, must be recalculated so that the various cursor regions may be redefined. The call to doChangeCursor redefines these regions for the new window location and copies the reference to one of them, depending on the current location of the mouse cursor, to the global variable gCursorRegion. (Note that this call to doChangeCursor is also required, for the

same reason, when a window is resized or zoomed.)

In the case of a resume event, `SetThemeCursor` is called to ensure that the cursor is set to the arrow shape.

In the case of a mouse-moved event (which occurs when the mouse cursor has moved outside the region whose reference is currently being passed in `WaitNextEvent`'s `mouseRgn` parameter), `doChangeCursor` is called to change the region passed in the `mouseRgn` parameter according to the current location of the mouse.

doMenuChoice

The purpose of the code prior to the switch is to cancel any cursor demonstrations that may be currently under way.

If the second of the first two animated cursor demonstrations is currently under way, `doReleaseAnimCursor` is called to release the memory associated with that animated cursor.

If either of the first two animated cursor demonstrations is currently under way, `SetThemeCursor` is called to set the cursor shape to the arrow shape and `WaitNextEvent`'s sleep parameter is then set to the maximum possible value.

If the third animated cursor demonstration is currently under way, the function `doAnimatedCursorOSX` is called to terminate the Mac OS X wait cursor.

If the cursor shape changing demonstration is currently under way, `gCursorRegionsActive` is set to false, the region containing the current cursor region is disposed of and the associated global variable is set to `NULL`, thus defeating the generation of mouse-moved events.

Note that, if the user chooses the About… item in the Mac OS 8/9 Apple or Mac OS X Application menu, `doAboutDialog` is called.

doOffScreenGWorld1

`doWithoutOffScreenGWorld` is the first demonstration.

Draw in Window

As a prelude for what is to come, the function `doDrawStuff` is called to repeatedly paint some shapes in the window.

Draw in Offscreen Graphics Port and Copy to Window

The call to `GetGWorld` saves the current graphics world-that is, the current graphics port and the current device.

The call to `NewGWorld` creates an offscreen graphics world. The `gworldPortPtr` parameter receives a pointer to the offscreen graphics world's graphics port. 0 in the `pixelDepth` parameter means that the offscreen world's pixel depth will be set to the deepest device intersecting the rectangle passed as the `boundsRect` parameter. This rectangle becomes the offscreen port's port port rectangle, the offscreen pixel map's bounding rectangle, and the offscreen device's bounding rectangle. `NULL` in the `cTable` parameter causes the default color table for the pixel depth to be used. The `aGDevice` parameter is set to `NULL` because the `noNewDevice` flag is not set. 0 in the flags parameter means that no flags are set.

The call to `SetGWorld` sets the graphics port pointed to by `gworldPortPtr` as the current graphics port. (When the first parameter is a `GWorldPtr`, the current device is set to the device attached to the offscreen world and the second parameter is ignored.)

`GetGWorldPixMap` gets a handle to the offscreen pixel map and `LockPixels` is called to prevent the base address of the pixel image from being moved when the pixel image is drawn into or copied from.

The call to `EraseRect` clears the offscreen graphics port before the function `doGWorldDrawing` is called to draw some graphics in the offscreen port.

With the drawing complete, the call to `SetGWorld` sets the (saved) window's graphics port as the current port and the saved device as the current device.

The next two lines establish the source and destination rectangles (required by the forthcoming call to `CopyBits`) as equivalent to the offscreen graphics world and window port rectangles respectively. The calls to `RGBForeColor` and `RGBBackColor` set the foreground and background colors to black and white respectively, which is required to ensure that the `CopyBits` call will produce predictable results in the color sense.

The `CopyBits` call copies the image from the offscreen world to the window. The call to `QDError` checks for any error resulting from the last QuickDraw call (in this case, `CopyBits`).

`UnlockPixels` unlocks the offscreen pixel image buffer and `DisposeGWorld` deallocates all of the memory previously allocated for the offscreen graphics world.

doOffScreenGWorld2

`doWithoutOffScreenGWorld` demonstrates the use of `CopyDeepMask` to copy a source pixel map to a destination pixel map using a pixel map as a mask, and clipping the copying operation to a designated region. Because mask pixel maps cannot come from the screen, an offscreen graphics world is created for the mask.

The first block loads a `'PICT'` resource and draws the picture in the window.

The current graphics world is then saved and an offscreen graphics world the same size as the drawn picture is created. The offscreen graphics port is set as the current port, the pixel map is locked, and the offscreen port is erased.

The second call to `GetPicture` loads the `'PICT'` resource representing the mask and `DrawPicture` is called to draw the mask in the offscreen port.

`SetGWorld` is then called again to make the window's graphics port the current port. The mask is then also drawn in the window next to the source image so that the user can see a copy of the mask in the offscreen graphics port.

The next two blocks define two regions, one containing an oval and one a rounded rectangle. The references to these regions will be passed in the `maskRgn` parameter of two separate calls to `CopyDeepMask`.

Before the calls to `CopyDeepMask`, the foreground and background colors are set to black and white respectively so that the results of the copying operation, in terms of color, will be predictable.

The for loop causes the source image to be copied to two locations in the window using a different mask region and Boolean source mode for each copy. The first time `CopyDeepMask` is called, the oval-shaped region is passed in the `maskRgn` parameter and the source mode `srcCopy` is passed in the mode parameter. The second time `CopyDeepMask` is called, the round rectangle-shaped region and `srcOr` are passed.

`QDError` checks for any error resulting from the last QuickDraw call (in this case, `CopyDeepMask`).

In the clean-up, `UnlockPixels` unlocks the offscreen pixel image buffer, `DisposeGWorld` deallocates all of the memory previously allocated for the offscreen graphics world, and the memory allocated for the picture resources and regions is released. Note that, because the pictures are resources obtained via `GetPicture`, `ReleaseResource`, rather than `KillPicture`, is used.

doPicture

`doPicture` demonstrates the recording and playing back of a picture.

Define Picture Rectangle and Set Clipping Region

The window's port rectangle is copied to a local `Rect` variable. This rectangle is then made equal to the left half of the port rectangle, and then inset by 10 pixels all round. This is the picture rectangle

The clipping region is then set to be the equivalent of this rectangle. (Before this call, the clipping region is very large. In fact, it is as large as the coordinate plane. If the clipping region is very large and you scale a picture while drawing it, the clipping region can become invalid when `DrawPicture` scales the clipping region, in which case the picture will not be drawn.)

Set Up OpenCPicParams Structure

This block assigns values to the fields of an `OpenCPicParams` structure. These specify the previously defined rectangle as the bounding rectangle, and 72 pixels per inch resolution both horizontally and vertically. The version field should always be set to -2.

Record Picture

`OpenCPicture` initiates the recording of the picture definition. The address of the `OpenCPicParams` structure is passed in the `newHeader` parameter.

The picture is then drawn. Lines, rectangles, round rectangles, ovals, wedges, and text are drawn in random colors, and sizes.

Stop Recording, Draw Picture, Restore Saved Clipping Region

The call to `ClosePicture` terminates picture recording and the call to `DrawPicture` draws the picture by "playing back" the "recording" stored in the specified Picture structure.

The call to `SetClip` restores the saved clipping region and `DisposeRgn` frees the memory associated with the saved region.

Display Some Information from the Pictinfo Structure

The call to `GetPictInfo` returns information about the picture in a picture information structure. Information in some of the fields of this structure is then drawn in the right side of the window.

Release Memory Occupied by Picture Structure

The call to `KillPicture` releases the memory occupied by the Picture structure.

doCursor

`doCursor`'s chief purpose is to assign true to the global variable `gCursorRegionsActive`, which will cause the function `doChangeCursor` to be called from within the main event loop provided the application is not in the background. In addition, it draws some rectangles in the window which visually represent to the user some cursor regions which will later be established by the `doChangeCursor` function.

The last two lines sets the `gCursorRegionsActive` flag to true and create an empty region for the last parameter of the `WaitNextEvent` call in the main event loop. A reference to a cursor region will be copied to `gCursorRegion` in the function `doChangeCursor`.

doChangeCursor

`doChangeCursor` is called whenever a mouse-moved event is received and after the window is dragged.

The first four lines create new empty regions to serve as the regions within which the cursor shape will be changed to, respectively, the arrow, I-beam, cross, and plus shapes.

The `SetRectRgn` call sets the arrow cursor region to, initially, the boundaries of the coordinate plane. The next five lines establish a rectangle equivalent to the window's port rectangle and change this rectangle's coordinates from local to global coordinates so that the regions calculated from it will be in the required global coordinates. The call to `InsetRect` insets this rectangle by 40 pixels all round and the call to `RectRgn` establishes this as the I-beam region. The call to `DiffRgn`, in effect, cuts the rectangle represented by the I-beam region from the arrow region, leaving a hollow arrow region.

The next six lines use the same procedure to establish a rectangular hollow region for the cross cursor and an interior rectangular region for the plus cursor. The result of all this is a rectangular plus cursor region in the center of the window, surrounded by (but not overlapped by) a hollow rectangular cross cursor region, this surrounded by (but not overlapped by) a hollow rectangular I-beam cursor region, this surrounded by (but not overlapped by) a hollow rectangular arrow cursor region the outside of which equates to the boundaries of the coordinate plane.

The call to `GetGlobalMouse` gets the point, in global coordinates, representing the mouse's current position.

The next task is to determine the region in which the cursor is currently located. The calls to `PtInRgn` are made for that purpose. Depending on which region is established as the

region in which the cursor in currently located, the cursor is set to the appropriate shape and the reference to that region is copied to the global variable passed in `WaitNextEvent`'s `mouseRgn` parameter.

That accomplished, the last four lines deallocate the memory associated with the regions created earlier in the function.

doAnimatedCursor1

`doAnimatedCursor1` responds to the user's selection of the Animated Cursor 1 item in the Demonstration menu.

In this first animated cursor demonstration, the Appearance Manager function `SetAnimatedThemeCursor` will be used in the function `doIncrementAnimCursor` to increment the cursor frame. As preparatory measures, an appropriate frame change tick interval is assigned to `gAnimCursTickInterval`, the sleep parameter in the `WaitNextEvent` call is set to the same value (causing null events to be generated at that tick interval), and `gAnimCurs1Active` is set to true so that `doIncrementAnimCursor` will be called from the `doIdle` function.

doAnimatedCursor2

`doAnimatedCursor2` responds to the user's selection of the Animated Cursor 2 item in the Demonstration menu.

In this second animated cursor demonstration, functions are utilized to retrieve `'acur'` and `'CURS'` resources, animate the cursor, and deallocate the memory associated with the animated cursor when the cursor is no longer required. `DoAnimatedCursor2`'s major role is simply to call `doGetAnimCursor` with a "beach-ball" `'acur'` resource as a parameter.

After the screen has been cleared, the resource ID of the "beach-ball" `'acur'` resource is assigned to the variable used as the first parameter in the later call to `doGetAnimCursor`. The next line assigns a value to the second parameter in the `doGetAnimCursor` call. This value controls the frame rate of the cursor, that is, the number of ticks which must elapse before the next frame (cursor) is displayed. (The best frame rate depends on the type of animated cursor used.)

If the call to `doGetAnimCursor` is successful, the sleep parameter in the `WaitNextEvent` call is set to the same ticks value as that used to control the cursor's frame rate (causing null events to be generated at that tick interval), and the flag `gAnimCurs2Active` is set to true so that `doIncrementAnimCursor` will be called from the `doIdle` function.

If the call to `getAnimCursor` fails, `doAnimCursor` simply plays the system alert sound and returns.

doGetAnimCursor

`doGetAnimCursor` retrieves the data in the specified `'acur'` resource and stores it in an `animCurs` structure, retrieves the 'CURS' resources specified in the `'acur'` resource and assigns the reference s to the resulting Cursor structures to elements in an array in the `animCurs` structure, establishes the frame rate for the cursor, and sets the starting frame number.

`GetResource` is called to read the `'acur'` resource into memory and return a handle to the resource. The handle is cast to type `animCursHandle` and assigned to the global variable `gAnimCursHdl` (a handle to a structure of type `animCurs`, which is identical to the structure of an `'acur'` resource). If this call is not successful (that is, `GetResource` returns `NULL`), the function will simply exit, returning false. If the call is successful, `noError` is set to true before a while loop is entered. This loop will cycle once for each of the `'CURS'` resources specified in the `'acur'` resource, assuming that `noError` is not set to false at some time during this process.

The ID of each cursor is stored in the high word of the specified element of the `frame[]` field of the `animCurs` structure. This is retrieved. The cursor ID is then used in the call to `GetCursor` to read in the resource (if necessary) and assign the handle to the resulting 68-byte Cursor structure to the specified element of the `frame[]` field of the `animCurs` structure. If this pass through the loop was successful, the array index is incremented; otherwise, `noError` is set to false, causing the loop and the function to exit.

The first line within the if block assigns the ticks value passed to `doGetAnimCursor` to a global variable that will be utilized in the function `doIncrementAnimCursor`. The next line assigns the number of ticks since system startup to another variable which will also be utilized in the function `doIncrementAnimCursor`. The third line sets the starting frame number.

At this stage, the animated cursor has been initialized and `doIdle` will call `doIncrementAnimCursor` whenever null events are received.

doIncrementAnimCursor

`doIncrementAnimCursor` is called whenever null events are received.

The first line assigns the number of ticks since system startup to newTick. The next line checks whether the specified number of ticks have elapsed since the previous call to `doIncrementAnimCursor`. If the specified number of ticks have not elapsed, the function simply returns. Otherwise, the following occurs:

- If the first animated cursor demonstration is under way, the Appearance Manager function `SetThemeAnimatedCursor` is called to increment the theme-compliant cursor frame.
- If the second animated cursor demonstration is under way, `SetCursor` sets the cursor shape to that represented by the handle stored in the specified element of the `frame[]` field of the `animCurs` structure. This line also increments the frame counter field (`whichFrame`) of the `animCurs` structure. If `whichFrame` has been incremented to the last cursor in the series, the frame counter is reset to 0.

The last line retrieves and stores the tick count at exit for use at the first line the next time the function is called.

doReleaseAnimCursor

`doReleaseAnimCursor` deallocates the memory occupied by the Cursor structures and the `'acur'` resource.

Recall that `doReleaseAnimCursor` is called when the user clicks in the menu bar and that, at the same time, the `gAnimatedCursor1Active` and `gAnimatedCursor2Active` flags are set to false, the cursor is reset to the standard arrow shape, and `WaitNextEvent`'s sleep parameter is reset to the maximum possible value.

doAnimatedCursorOSX

`doAnimatedCursorOSX` utilizes the function `QDDisplayWaitCursor` to turn the Mac OS X "spinning wheel" cursor on and off. This function is only available on OS X, so the stub library `CarbonFrameworkLib` has been added to the CodeWarrior project.

doIcon

`doIcon` demonstrates the drawing of icons in a window using `PlotIconID`, `PlotIconSuite`, and `PlotCIcon`.

PlotIconID With Transforms

This block uses the function `PlotIconID` to draw an icon from an icon family with the specified ID fifteen times, once for each of the fifteen available transform types. `PlotIconID` automatically chooses the appropriate icon resource from the icon suite depending on the specified destination rectangle and the bit depth of the current device.

GetIconSuite and PlotIconSuite

This block uses `GetIconSuite` to get an icon suite comprising only the `'ICN#'`, `'icl4'`, and `'icl8'` resources from the icon family with the specified resource ID. `PlotIconSuite` is then called three times to draw the appropriate icon within destination rectangles of three different sizes. `PlotIconSuite` automatically chooses the appropriate icon resource from the icon suite depending on the specified destination rectangle and the bit depth of the current device. `PlotIconSuite` also expands the icon to fit the last two destination rectangles.

GetCIcon and PlotCIcon

This block uses `GetCIcon` to load the specified `'cicn'` resource and `PlotCIcon` to draw the color icon within destination rectangles of three different sizes. `PlotCIcon` expands the 32-by 32-pixel icon to fit the last two destination rectangles.

doAboutDialog

`doAboutDialog` is called when the user chooses the About… item in the Apple menu.

`GetNewDialog` creates a modal dialog. The dialog's item list contains a picture item, which fills the entire dialog window.

The call to `ModalDialog` means that the dialog will remain displayed until the user clicks somewhere within the dialog, at which time `DisposeDialog` is called to dismiss the dialog and free the associated memory. A dialog rather than an alert is used to obviate the need for a button for dismissing the dialog.

Chapter 14. More on Controls

IN THIS CHAPTER

- Introduction
- Control Descriptions
- Small Versions of Controls
- Idle Processing
- Defining Your Own Key Filter Function
- Defining Your Own Edit Text Validation Function
- Defining Your Own User Pane Functions
- Main Constants, Data Types and Functions
- Demonstration Program Controls3 Listing
- Demonstration Program Controls3 Comments

Introduction

Chapter 7 introduced the subject of controls and addressed the basic controls (push buttons, check boxes, radio buttons, scroll bars, and pop-up menu buttons), primary group boxes (text title variant) and user panes. This chapter revisits group boxes and user panes and addresses the many remaining controls.

The controls addressed in this chapter are as follows:

Bevel button	Image well	Tab
Group boxes	Clock	Progress and relevance bar
Picture	Icon	Window header
Separator line	Pop-up arrow	Radio group
Scrolling text box	Disclosure button	Edit Unicode text
Edit text	Little arrows	Placard
Slider	Disclosure triangle	Chasing arrows
Round button	Static text	User pane

The data browser control is not addressed in this book. The one remaining control (the list box) is addressed in Chapter 22.

The Progress bar will be described in this chapter and the indeterminate variant demonstrated in the associated demonstration program. However, because the use of the determinate variant is ordinarily associated with the matter of scanning for a Command-period event, the demonstration of that particular variant has been held over to the demonstration program associated with Chapter 25.

Preamble-Review of Control Basics

Recall from Chapter 7 that

- Control definition functions (**CDEF**s) determine the appearance and behavior of controls.
- Two new features introduced with Mac OS 8 were **embedding** and **latency.** Embedding hierarchies have implications for drawing order and hit testing.
- Another feature introduced with Mac OS 8 was read and write access to the various

attributes of a control. Each piece of information that a particular CDEF allows access to is referenced by a control data tag. **Control data tag constants** are passed in the `inTagName` parameter of the getter and setter functions `SetControlData` and `GetControlData`.

- `FindControl` is used to determine whether a mouse-down event occurred in a control and, if so, which control. `FindControl` returns a **control part code** identifying the part of the control in which the mouse-down occurred. `kControlNoPart` (`0`) is returned if the mouse-down occurred where no enabled control exists. `TrackControl` or `HandleControlClick` are used to handle user interaction with a control as long as the user holds the mouse button down. These two functions return `kControlNoPart` if the cursor is outside the control or control part when the mouse button is released or the relevant control part code if the user releases the mouse button while the cursor is still inside the control or control part.
- The font for any control can be set independently of the system or window font.
- The pop-up menu button differs from the other basic controls in that the control's initial, minimum, and maximum values do not actually represent initial, minimum, and maximum values as such. For example, the minimum value field of the `'CNTL'` resource for a pop-up button is used to specify the resource ID of the `'MENU'` resource utilized by the control.

The use of the initial, minimum, and maximum fields for purposes other than control values as such also applies to most of the controls addressed in this chapter.

More on Embedding

As stated in Chapter 8, if the `kDialogFlagsUseControlHierarchy` feature bit is set in a dialog's `'dlgx'` resource, the Dialog Manager creates a root control in the dialog and establishes an embedding hierarchy.

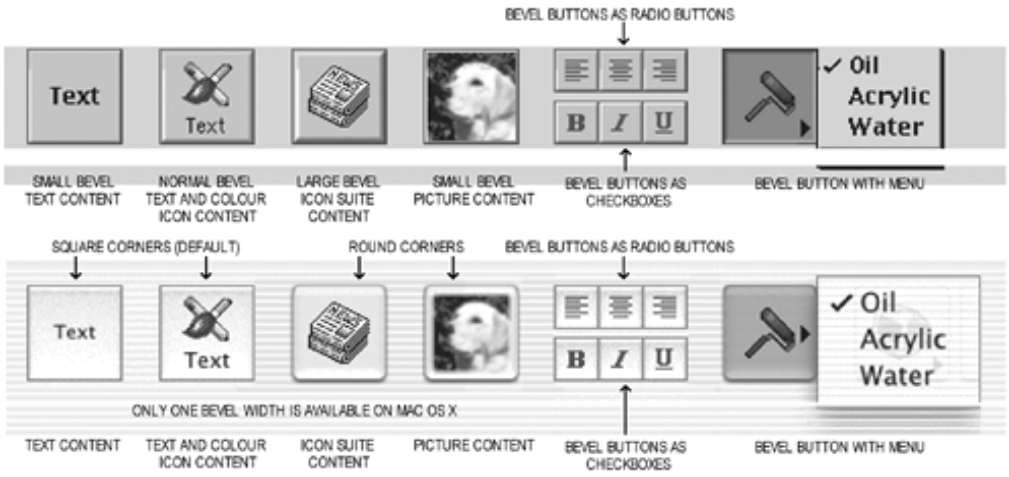
The Dialog Manager uses `AutoEmbedControl` to position dialog items in an embedding hierarchy based on both visual containment and the order of the items in the item list. As items are added to a dialog during creation, controls that already exist in the window will be containers for new controls if they both visually contain the control and are themselves embedder controls. For this reason, you should place the largest embedder controls at the beginning of the item list. As an example, the Dialog Manager will automatically embed radio buttons in a group box if the radio buttons visually "fit" inside the group box and the group box precedes the radio buttons in the item list.

Control Descriptions

Bevel Buttons

A bevel button is a rectangular control with a beveled edge that can include text, an icon, a picture, or a combination of text and an icon or picture. Bevel buttons can be made to behave in a number of different ways: They can behave like push buttons; in sets, they can behave like radio buttons or checkboxes; if a menu is attached, they behave like pop-up menu buttons. Typical bevel buttons are shown in Fig 1.

Figure 1. TYPICAL BEVEL BUTTONS



Note that, on Mac OS X, bevel button have square corners by default but can be made rounded using the function `SetControlData` with the `kControlBevelButtonKindTag` tag (see "Other Control Data Tag Constants" later in this chapter).

Variants and Control Definition IDs

The bevel button CDEF resource ID is 2. The six available variants and their control definition IDs are as follows:

Variant	*Var Code*	*Control Definition ID*
With small (2 pixel wide) bevel. (Pop-up menu, if any, below.)	`0`	`32 kControlBevelButtonSmallBevelProc`
With medium (3 pixel wide) bevel. (Pop-up menu, if any, below.)	`1`	`33 kControlBevelButtonNormalBevelProc`
With large (4 pixel wide) bevel. (Pop-up menu, if any, below.)	`2`	`34 kControlBevelButtonLargeBevelProc`

With small (2 pixel wide) bevel. Pop-up menu on right.	4	36	kControlBevelButtonSmallBevelProc + kControlBevelButtonMenuOnRight
With medium (3 pixel wide) bevel. Pop-up menu, if any, on right.	5	37	kControlBevelButtonNormalBevelProc + kControlBevelButtonMenuOnRight
With large (4 pixel wide) bevel. Pop-up menu, if any, on right.	6	38	kControlBevelButtonLargeBevelProc + kControlBevelButtonMenuOnRight

Note that, while three bevel sizes are available on Mac OS 8/9, bevel button will always be displayed with one bevel size on Mac OS X regardless of which of the first three variants is used.

For bevel buttons containing text, if the constant `kControlUsesOwningWindowsFontVariant` is added to the variation code, the text will appear in the window's font.

Control Values

The following lists the initial, minimum, and maximum value settings for bevel buttons:

Control Value	*Content*
Initial	If you wish to attach a menu, the menu ID. If no menu, 0.
Minimum	High byte specifies behavior (see "Bevel Button Behavior Constants," and "Bevel Button Menu Constants"). Low byte specifies content type (see "Bevel Button and Image Well Content Type Constants").
Maximum	Resource ID of bevel button's content if resource-based.

Note that bevel buttons have two values: the value of the bevel button and, if a menu has been attached, the value of the menu.

Bevel Button Behavior Constants

The following **bevel button behavior constants** apply to the high byte of a bevel button's minimum value:

Constant	*Value*	*Description*
`kControlBehaviorPushbutton`	`0`	Button pops up after being clicked. This constant is used when push button behavior is required.
`kControlBehaviorToggles`	`0x0100`	Button toggles state automatically when clicked. This constant is used when checkbox or radio button behavior is required.
`kControlBehaviorSticky`	`0x0200`	When clicked, button stays down until application sets control's value to 0. This behavior is useful in tool palettes.

kControlBehaviorOffsetContents	0x8000	Button contents are offset (one pixel down and to the right) when button is clicked. (Some users consider that this behavior gives a bevel button a more realistic "feel.")

Bevel Button Behavior Constants (Menus)

The following bevel button behavior constants (menus) apply to the high byte of a bevel button's minimum value:

Constant	*Value*	*Description*
kControlBehaviorSingleValueMenu	0	Menu contains commands, not choices, and should not be marked with a checkmark. Overrides the kControlBehaviorMultiValueMenu bit.
kControlBehaviorMultiValueMenu	0x4000	Menus are multi-valued. The user can toggle entries in the button's menu and have multiple items checked. The button does not maintain the menu value as it normally would. When this bit is set, the menu value accessed using GetControlData with the kControlBevelButtonMenuValueTag will return the value of the last menu item chosen.

Button and Image Well Content Type Constants

The following **button and image well content type constants** apply to the low byte of a bevel button's minimum value:

Constant	*Value*	*Description*
kControlContentTextOnly	0	Content type is text only.
kControlContentIconSuiteRes	1	Content type is an icon suite resource ID.
kControlContentCIconRes	2	Content type is a color icon resource ID.
kControlContentPictRes	3	Content type is a picture resource ID.
kControlContentIconSuiteHandle	129	Content type is an icon suite handle.
kControlContentCIconHandle	130	Content type is a color icon handle.
kControlContentPictHandle	131	Content type is a picture handle.
kControlContentIconRef	132	Content type is an icon reference.

You can also use these constants in the contentType field of the **button content info structure.** You can then pass a pointer to this structure in the inBuffer parameter of GetControlData and SetControlData to get and set the resource ID (for resource-based content), handle (for handle-based content), or reference (for reference-based content) of a color icon, icon suite, picture, or icon reference in a bevel button.

> **Note**
>
> Note that resource-based content is owned by the control, while handle-based content is owned by you. The control definition function will not dispose of handle-based content. If you replace handle-based content with resource-based content on the fly, you must dispose of the handle properly to avoid a memory leak.

Control Data Tag Constant-Content Type

The control data tag constant relevant to content type in bevel buttons is as follows:

Control Data Tag Constant	*Meaning and Data Type Returned or Set*

`kControlBevelButtonContentTag`	Gets or sets a bevel button's content type for drawing. (See "Bevel Button and Image Well Content Type Constants." See also "The Bevel Button and Image Well Structure" section later in this chapter.) **Data type returned or set:** `ControlButtonContentInfo` structure

Bevel Button Alignment and Placement Constants and Associated Control Data Tag Constants

By calling `SetControlData` with certain control data tag constants, and with certain constants passed in the `inData` parameter, you can specify the alignment of icons, pictures, and text in a bevel button, and you can specify the placement of text in relation to an icon or picture. By calling `GetControlData` with these constants you can also ascertain alignment and placement.

Bevel Button Graphic Alignment Constants

The following constants are used to specify the alignment of icon suites, color icons, and pictures:

Constant	*Value*	*Description*
`kControlBevelButtonAlignSysDirection`	`-1`	Graphic is aligned according to the system default script direction (only left or right).
`kControlBevelButtonAlignCenter`	`0`	Graphic is aligned center.
`kControlBevelButtonAlignLeft`	`1`	Graphic is aligned left.
`kControlBevelButtonAlignRight`	`2`	Graphic is aligned right.
`kControlBevelButtonAlignTop`	`3`	Graphic is aligned top.
`kControlBevelButtonAlignBottom`	`4`	Graphic is aligned bottom.
`kControlBevelButtonAlignTopLeft`	`5`	Graphic is aligned top left.
`kControlBevelButtonAlignBottomLeft`	`6`	Graphic is aligned bottom left.
`kControlBevelButtonAlignTopRight`	`7`	Graphic is aligned top right.
`kControlBevelButtonAlignBottomRight`	`8`	Graphic is aligned bottom right.

Bevel Button Text Alignment Constants

The following constants are used to specify the alignment of text:

Constant	*Value*	*Description*
`kControlBevelButtonAlignTextSysDirection`	`0`	Text is aligned according to the current script direction (left or right).
`kControlBevelButtonAlignTextCenter`	`1`	Text is aligned center.
`kControlBevelButtonAlignTextFlushRight`	`-1`	Text is aligned flush right.
`kControlBevelButtonAlignTextFlushLeft`	`-2`	Text is aligned flush left.

Bevel Button Text Placement Constants

The following constants are used to specify the placement of text in relation to an icon suite, color icon, or picture:

Constant	*Value*	*Description*
`kControlBevelButtonPlaceSysDirection`	`-1`	Text is placed according to the system default script direction.

`kControlBevelButtonPlaceNormally`	`0`	Text is centered.
`kControlBevelButtonPlaceToRightOfGraphic`	`1`	Text is placed to the right of the graphic.
`kControlBevelButtonPlaceToLeftOfGraphic`	`2`	Text is placed to the left of the graphic.
`kControlBevelButtonPlaceBelowGraphic`	`3`	Text is placed below the graphic.
`kControlBevelButtonPlaceAboveGraphic`	`4`	Text is placed above the graphic.

Control Data Tag Constants-Alignment and Placement

The control data tag constants relevant to the alignment and placement of graphics and text in bevel buttons are as follows:

Control Data Tag Constant	*Meaning and Data Type Returned or Set*
`kControlBevelButtonTextAlignTag`	Gets or sets the alignment of text. (See "Bevel Button Text Alignment Constants.") **Data type returned or set:** `ControlButtonTextAlignment`
`kControlBevelButtonGraphicAlignTag`	Gets or sets the alignment of graphics in relation to any text the button may contain. (See "Bevel Button Graphic Alignment Constants.") **Data type returned or set:** `ControlButtonGraphicAlignment`
`kControlBevelButtonTextPlaceTag`	Gets or sets the placement of text. (See "Bevel Button Text Placement Constants.") **Data type returned or set:** `ControlButtonTextPlacement`
`kControlBevelButtonTextOffsetTag`	Gets or sets the number of pixels that text is offset from the button's left or right edge. This is used with left, right, or system justification, but it is ignored when the text is center aligned. **Data type returned or set:** `SInt16`
`kControlBevelButtonGraphicOffsetTag`	Gets or sets the horizontal and vertical amounts that a graphic element is offset from the button's edges. Ignored when the graphic is specified to be centered on the button. **Data type returned or set:** `Point`

The Button Content Info Structure

As previously stated, you can use button and image well content type constants in the `contentType` field of the **button content info structure** and you can then pass a pointer to this structure in the `inBuffer` parameter of `GetControlData` and `SetControlData`. The button content info structure is as follows:

```
struct ControlButtonContentInfo
{
  ControlContentType  contentType;
  union {
    SInt16      resID;
    CIconHandle cIconHandle;
    Handle      iconSuite;
```

```
    IconRef     iconRef;
    PicHandle   picture;
    Handle      ICONHandle;
  }  u;
} ;
typedef struct ControlButtonContentInfo ControlButtonContentInfo;
typedef ControlButtonContentInfo *ControlButtonContentInfoPtr;
typedef ControlButtonContentInfo ControlImageContentInfo;
typedef ControlButtonContentInfo *ControlImageContentInfoPtr;
```

Field Descriptions

`contentType` Specifies the content type (see "Bevel Button and Image Well Content Type Constants") and determines which of the following fields are used.

`resID` If the content type specified in the `contentType` field is `kControlContentIconSuiteRes`, `kControlContentCIconRes`, or `kControlContentPictRes`, this field should contain the resource ID of a picture, color icon, or icon suite resource.

`cIconHandle` If the content type specified in the `contentType` field is `kControlContentCIconHandle`, this field should contain a handle to a color icon.

`iconSuite` If the content type specified in the `contentType` field is `kControlContentIconSuiteHandle`, this field should contain a handle to an icon suite.

`iconRef` If the content type specified in the `contentType` field is `kControlContentIconRef`, this field should contain an icon reference.

`picture` If the content type specified in the `contentType` field is `kControlContentPictHandle`, this field should contain a handle to a picture.

Other Control Data Tag Constants

The remaining control data tag constants relevant to bevel buttons are as follows:

Control Data Tag Constant	*Meaning and Data Type Returned or Set*
`kControlBevelButtonCenterPopUpGlyphTag`	Gets or sets the position of the pop-up arrow when a pop-up menu is attached. **Data type returned or set:** `Boolean`. If `true`, glyph is vertically centered on the right; if `false`, glyph is on the bottom right.
`kControlBevelButtonTransformTag`	Gets or sets a transform that is added to the standard transform. See the Icons section in Chapter 13.) **Data type returned or set:** `IconTransformType`
`kControlBevelButtonMenuValueTag`	Gets the menu value. (See "Bevel Button Menu Constants.") **Data type returned:** `SInt16`
`kControlBevelButtonMenuRefTag`	Gets the menu reference. **Data type returned:** `MenuRef`

`kControlBevelButtonLastMenuTag`	Gets the menu ID of the last menu selected in the submenu or main menu. **Data type returned:** `Boolean`
`kControlBevelButtonMenuDelayTag`	Gets or sets the delay (in number of ticks) before the menu is displayed. **Data type returned or set:** `SInt32`
`kControlBevelButtonScaleIconTag`	Gets or sets whether, when the proper icon size is unavailable, the icon should be scaled. For use only with icon suites or the `IconRef` data type. **Data type returned or set:** `Boolean`. If `true`, indicates that if an icon of the ideal size is not available a larger or smaller icon should be scaled. If `false`, no scaling should occur; instead, a smaller icon should be drawn or a larger icon clipped.
`kControlBevelButtonContentTag`	Sets bevel button content. **Data type reset:** `ButtonContentInfo`
`kControlBevelButtonOwnedMenuRefTag`	Sets the menu reference. (The control will dispose.) **Data type set:** `MenuRef`
`kControlBevelButtonKindTag`	Sets the button kind. **Data type returned or set:** `ThemeButtonKind` (`kThemeSmallBevelButton`, `kThemeMediumBevelButton`, `kThemeLargeBevelButton`, `kThemeRoundedBevelButton`).

With regard to the `kControlBevelButtonMenuDelayTag` constant, setting a delay before the menu is displayed in a bevel button with sticky behavior is useful for providing option sets in tool palettes. You can set up the bevel button so that

- If the user clicks it once, it simply turns on the function represented by the button.
- If the user presses it for longer than the user-set double-click time, it displays a pop-up menu which offers further options for that function.

Helper Functions

The following helper functions may be used in lieu of `SetControlData` and `GetControlData` with the relevant control data tags:

```
GetBevelButtonMenuValue
SetBevelButtonMenuValue
GetBevelButtonMenuHandle
GetBevelButtonContentInfo
SetBevelButtonContentInfo
SetBevelButtonTransform
SetBevelButtonGraphicAlignment
SetBevelButtonTextAlignment
SetBevelButtonTextPlacement
```

Control Part Codes

The following control part codes are relevant to bevel buttons:

Constant	*Value*	*Description*
`kControlNoPart`	`0`	For bevel buttons with a menu attached, this part code indicates that either the mouse was released outside the bevel button and menu or that the button was disabled.
`kControlMenuPart`	`2`	For bevel buttons with a menu attached, this part code indicates that the event occurred in a menu item of the bevel button.
`kControlButtonPart`	`10`	For bevel buttons with a menu attached, this part code indicates that the event occurred in the button but not in the attached menu.

Bevel Button States

Bevel buttons can exist in five active states and two disabled states. The active states are off, pressed (was off) on, pressed (was on), and mixed. The mixed state is used, where appropriate, when the bevel button is behaving as a checkbox or radio button. Disabled bevel buttons can be shown as off or on.

Programmatic Creation

Bevel button controls may be created programmatically using the function `CreateBevelButtonControl`:

```
OSStatus  CreateBevelButtonControl(WindowRef window,const Rect *boundsRect,
                                   CFStringRef title,ControlBevelThickness thickness,
                                   ControlBevelButtonBehavior behavior,
                                   ControlButtonContentInfoPtr info,SInt16 menuID,
                                   ControlBevelButtonMenuBehavior menuBehavior,
                                   ControlBevelButtonMenuPlacement menuPlacement,
                                   ControlRef *outControl);
```

Relevant constants are

Parameter	*Constants*

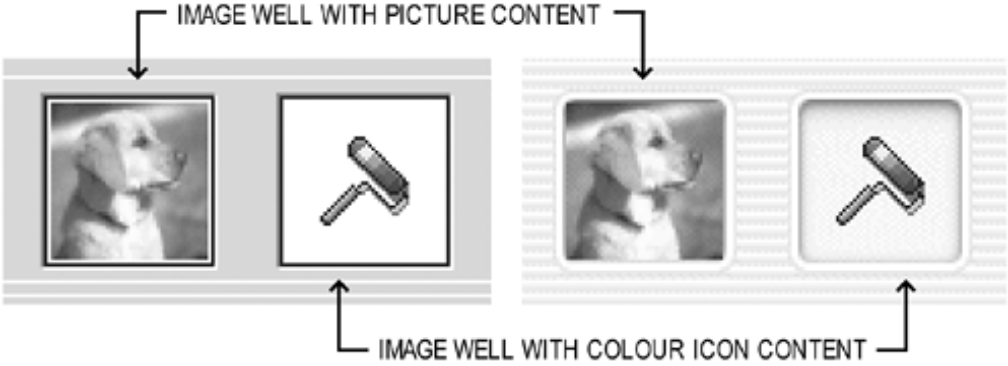
`thickness`	`kControlBevelButtonSmallBevel`
	`kControlBevelButtonNormalBevel`
	`kControlBevelButtonLargeBevel`

`behavior`	See "Bevel Button Behavior Constants."
`info.contentType`	See "Button and Image Well Content Type Constants."
`menuBehavior`	See "Bevel Button Behavior Constants (Menus)."
`menuPlacement`	`kControlBevelButtonMenuOnBottom` `kControlBevelButtonMenuOnRight`

Image Wells

Image wells can be used to display icons or pictures. They are controlled in much the same way as bevel buttons, but with fewer options and states. They should not be used in place of push buttons or bevel buttons. Typical image wells are shown in Fig 2.

Figure 2. TYPICAL IMAGE WELLS



Variants and Control Definition IDs

The image well CDEF resource ID is 11. The one available variant and its control definition IDs is as follows:

Variant	*Var Code*	*Control Definition ID*
Image well	`0`	`176` `kControlImageWellProc`

Control Values

Control Value	*Content*
Initial	Resource ID of image well's content (`'cicn'`, `'PICT'` or icon suite).
Minimum	Content type. (See "Bevel Button and Image Well Content Type Constants.") After the image well is created, reset to 0.
Maximum	Ignored. Reset to 2 after creation.

Bevel Button and Image Well Content Type Constants and The Button Content Info Structure

The button and image well content type constants (see "Bevel Buttons") apply to an image well's minimum value.

You can also use these constants in the `contentType` field of the button content info structure (see "Bevel Buttons"). You can then pass a pointer to this structure in the `inBuffer` parameter of `GetControlData` and `SetControlData` to get and set the resource ID (for resource-based content) or handle (for handle-based content) of a color icon, icon suite, or picture in an image well.

Control Data Tag Constants

Control Data Tag Constant	*Meaning and Data Type Returned or Set*
`kControlImageWellContentTag`	Gets or sets the content. (See "The Button Content Info Structure.") **Data type returned or set:** `ControlButtonContentInfo` structure
`kControlImageWellTransformTag`	Gets or sets a transform that is added to the standard transform. (See the "Icons" section in Chapter 13.) **Data type returned or set:** `IconTransformType`
`kControlImageWellIsDragDestinationTag`	**Data type returned or set:** `Boolean`

Helper Functions

The following helper functions may be used in lieu of `SetControlData` and `GetControlData` with the relevant control data tag constants:

```
GetImageWellContentInfo
SetImageWellContentInfo
SetImageWellTransform
```

Control Part Codes

Constant	*Value*	*Description*
`kControlImageWellPart`	`26`	Event occurred in an image well

Programmatic Creation

Image well controls may be created programmatically using the function `CreateImageWellControl`:

```
OSStatus  CreateImageWellControl(WindowRef window,const Rect *boundsRect,
                                 const ControlButtonContentInfo *info,
                                 ControlRef *outControl);
```

Relevant constants are

Parameter	*Constants*
`info.contentType`	See "Button and Image Well Content Type Constants."

Tab-Embedding Control

The tab control is an embedding control which provides a means of presenting information on multiple "pages." The user selects the desired "page" by clicking the appropriate tab.

A typical tab control is shown in Fig 3.

Figure 3. TAB CONTROL



The content area of a tab is known as a **pane.** Controls that are embedded within an individual pane should only affect the settings displayed in that pane. Controls whose effect is intended to be global (that is, their settings are intended to affect all the panes in a set of tabs) should be located outside the tab control.

The tab information (`'tab#'`) resource may be used to provide the tab names and icon suite IDs.

Variants and Control Definition IDs

The tab control CDEF resource ID is 8. The eight available variants and their control definition ID are follows:

Variant	*Var Code*	*Control Definition ID*
Large tab control, tabs at top.	`0`	`128` `kControlTabLargeNorthProc`

Small tab control, tabs at top.	1	129	kControlTabSmallNorthProc
Large tab control, tabs at bottom.	2	130	kControlTabLargeSouthProc
Small tab control, tabs at bottom.	3	131	kControlTabSmallSouthProc
Large tab control, tabs at right.	4	132	kControlTabLargeEastProc
Small tab control, tabs at right.	5	133	kControlTabSmallEastProc
Large tab control, tabs at left.	6	134	kControlTabLargeWestProc
Small tab control, tabs at left.	7	135	kControlTabSmallWestProc

Control Values

Control Value	*Content*
Initial	Resource ID of the 'tab#' resource you are using to hold tab information. Reset to the minimum setting after creation. A value of 0 indicates not to read a 'tab#' resource. (See "The Tab Information Structure.")
Minimum	Ignored. Reset to 1 after creation.
Maximum	Ignored. Reset to the number of individual tabs in the tab control after creation.

Control Data Tag Constants

Control Data Tag Constant	*Meaning and Data Type Returned or Set*
kControlTabContentRectTag	Gets the content rectangle. **Data type returned:** Rect
kControlTabEnabledFlagTag	Enables or disables a single tab. **Data type returned or set:** Boolean; if true, enabled; if false, disabled.
kControlTabFontStyleTag	Gets or sets the font and font style. **Data type returned or set:** ControlFontStyleRec
kControlTabInfoTag	Gets or sets information for a tab in a tab control. (See "The Tab Information Structure.") **Data type returned or set:** ControlTabInfoRec

Helper Functions

The following helper functions may be used in lieu of SetControlData and GetControlData with the relevant control data tags:

```
GetTabContentRect
SetTabEnabled
```

The Tab Information Structure

If you are not creating a tab control with a 'tab#' resource, you can call SetControlMaximum to set the number of tabs in a tab control and then call SetControlData with the kControlTabInfoTag to set the information for an individual tab in a

tab control. The tab information structure passed in the `SetControlData` call should be one of the following:

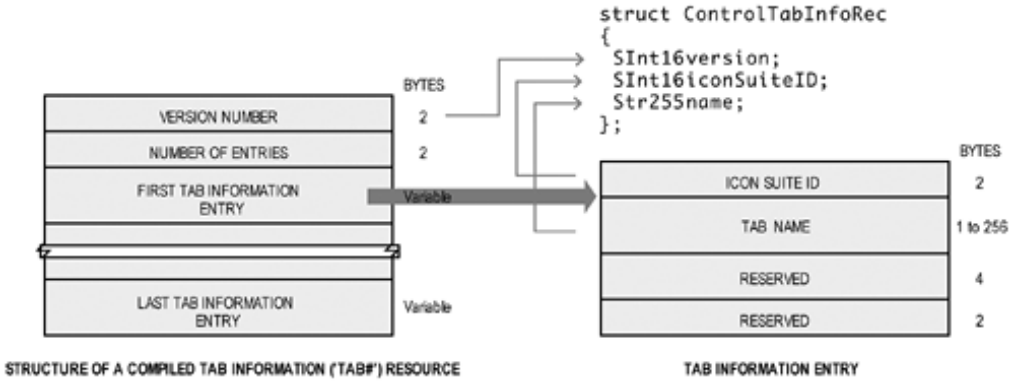
```
struct ControlTabInfoRec
{
  SInt16 version;      // Set to kControlTabInfoVersionZero.
  SInt16 iconSuiteID;  // Set to 0 for no icon.
  Str255 name;         // Title for tab label.
} ;
typedef struct ControlTabInfoRec ControlTabInfoRec;

struct ControlTabInfoRecV1
{
  SInt16      version;     // Set to kControlTabInfoVersionOne.
  SInt16      iconSuiteID; // Set to 0 for no icon.
  CFStringRef name;        // Title for tab label. (Should always be released)
} ;
typedef struct ControlTabInfoRecV1 ControlTabInfoRecV1;
```

The Tab Information Resource

You can use a tab information resource to specify the icon suite ID and name of each tab in a tab control. A tab information resource is a resource of type `'tab#'`. All tab information resources must have resource ID numbers greater than 127. The Control Manager uses the information you specify to provide additional information to the corresponding tab control. Fig 4 shows the structure of a compiled `'tab#'` resource.

Figure 4. STRUCTURE OF A COMPILED TAB INFORMATION ('tab#') RESOURCE



The following describes the fields of a compiled `'tab#'` resource and one of its tab information entries:

Field	*Description*
VERSION NUMBER	Version of the resource
NUMBER OF ENTRIES	The number of tab information entries in the resource
ICON SUITE ID	Icon suite resource ID
TAB NAME	The tab name

Programmatic Creation

Tab controls may be created programmatically using the function `CreateTabsControl`:

```
OSStatus  CreateTabsControl(WindowRef window,const Rect *boundsRect,ControlTabSize size,
                            ControlTabDirection direction,UInt16 numTabs,
                            const ControlTabEntry *tabArray,
                            ControlRef *outControl);
```

Relevant constants are

Parameter	*Constants*
`size`	`kControlTabSizeLarge` `kControlTabSizeSmall`

`direction`	`kControlTabDirectionNorth`
	`kControlTabDirectionSouth`
	`kControlTabDirectionEast`
	`kControlTabDirectionWest`

The `tabArray` parameter of takes a pointer to an array of **tab entry structures:**

```
struct ControlTabEntry
{
  ControlButtonContentInfo *icon;
  CFStringRef               name;
  Boolean                   enabled;
} ;
typedef struct ControlTabEntry ControlTabEntry;
```

Edit Text

Edit text controls are rectangular areas in which the user enters text. Edit text controls support keyboard focus, and a password entry variant is available. Fig 5 shows a typical edit text control.

Figure 5. EDIT TEXT CONTROL



Edit text controls can have a key filter function attached to filter key strokes or modify them on return.

Variants and Control Definition IDs

The edit text control CDEF resource ID is 17. The three available variants and their control definition IDs are as follows:

Variant	*Var Code*	*Control Definition ID*
Edit text control.	`0`	`272` `kControlEditTextProc`
Edit text control for passwords. This control is supported by the Script Manager. Password text can be accessed via the `kEditTextPasswordTag` constant. (See "Control Data Tag Constants.")	`2`	`274` `kControlEditTextPasswordProc`
Edit text control for inline input. This control supports 2-byte script systems.	`4`	`276` `kControlEditTextInlineInputProc`

Control Values

Control Value	*Content*
Initial	Reserved. Set to 0.
Minimum	Reserved. Set to 0.

Maximum	Reserved. Set to 0.

Main Control Data Tag Constants

Control Data Tag Constant	*Meaning and Data Type Returned or Set*
`kControlEditTextTextTag`	Gets or sets text. **Data type returned or set:** character buffer
`kControlEditTextTEHandleTag`	Gets a handle to the text edit structure. **Data type returned or set:** `TEHandle`
`kControlEditTextSelectionTag`	Gets or sets the selection. **Data type returned or set:** `ControlEditTextSelectionRec` structure. (See "The Edit Text Selection Structure.")
`kControlEditTextPasswordTag`	Gets clear password text, that is, the text of the actual password typed, not the bullet text. **Data type returned or set:** character buffer
`kControlKeyFilterTag`	Gets or sets a key filter function. **Data type returned or set:** `ControlKeyFilterUPP`
`kControlEditTextStyleTag`	Gets or sets the font style. **Data type returned or set:** `ControlFontStyleRec`
`KControlEditTextLockedTag`	Gets or sets whether the text is currently editable. **Data type returned or set:** `Boolean`. If `true`, text is locked. If `false`, text is editable
`kControlEditTextValidationProcTag`	Gets or sets a universal procedure pointer to a callback function which can be used to validate editable text after an operation that changes the text, such as a cut or paste. **Data type returned or set:** `ControlEditTextValidationUPP`
`kControlEditTextCFStringTag`	Gets or sets text. (Mac OS X only.) **Data type returned or set:** `CFStringRef`

Control Part Codes

Constant	*Value*	*Description*

`kControlEditTextPart`	5	Event occurred in an Edit text control.

The Edit Text Selection Structure

You can pass a pointer to the edit text selection structure to `GetControlData` and `SetControlData` to access and set the current selection range in an edit text control. An edit text selection structure is of type `ControlEditTextSelectionRec`:

```
struct ControlEditTextSelectionRec
{
  SInt16 selStart;
  SInt16 selEnd;
} ;
typedef struct ControlEditTextSelectionRec ControlEditTextSelectionRec;
typedef ControlEditTextSelectionRec *ControlEditTextSelectionPtr;
```

Field Descriptions

`selStart` Beginning of the edit text selection.

`selEnd` End of the edit text selection.

Programmatic Creation

Edit text controls may be created programmatically using the function `CreateEditTextControl`:

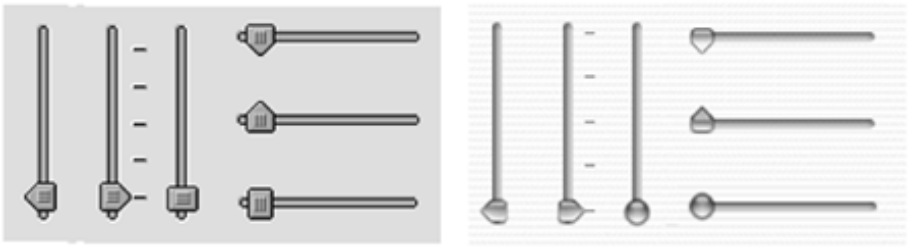
```
OSStatus  CreateEditTextControl(WindowRef window,const Rect *boundsRect,CFStringRef text,
                                Boolean isPassword,Boolean useInlineInput,
                                const ControlFontStyleRec *style,ControlRef *outControl);
```

Sliders

A slider control comprises a **slider bar** and an **indicator.** The user can drag the indicator to set a new value within the range represented by the slider bar.

Sliders can be oriented horizontally or vertically, and the indicator can point in any direction. The indicator can also be nondirectional if required. Typical sliders are shown in Fig 6.

Figure 6. SLIDERS



A slider can, optionally, display tick marks, and you can specify the number of tick marks required. If you specify tick marks, you should ensure that they are labeled.

A live feedback variant is available. This variant continually updates the value of the control as the indicator is dragged, as opposed to the standard behavior of updating the value only when the mouse button is released.

Variants and Control Definition IDs

The slider CDEF resource ID is 3. The ten available variants and their control definition IDs are as follows:

Variant	*Var Code*	*Control Definition ID*
Slider. If the slider is horizontal, the indicator points down, and if the slider is vertical, the indicator points right.	0	48 kControlSliderProc
Slider with live feedback. The value of the control is updated automatically by the Control Manager before your action function is called. If no application-defined action function is supplied, the slider draws ghosted image of the indicator as the user moves it.	1	49 kControlSliderProc + kControlSliderLiveFeedback
Slider with tick marks. The control rectangle must be large enough to include the tick marks.	2	50 kControlSliderProc + kControlSliderHasTickMarks
Slider with live feedback and tick marks.	3	51 kControlSliderProc + kControlSliderLiveFeedback + kControlSliderHasTickMarks
Slider with indicator reversed. If the slider is horizontal, the indicator points up, and if the slider is vertical, the indicator points left.	4	52 kControlSliderProc + kControlSliderReverseDirection
Slider with live feedback and indicator reversed.	5	53 kControlSliderProc+ kControlSliderLiveFeedback + kControlSliderReverseDirection
Slider with tick marks and indicator reversed.	6	54 kControlSliderProc + kControlSliderHasTickMarks + kControlSliderReverseDirection
Slider with live feedback, tick marks and indicator reversed.	7	55 kControlSliderProc + kControlSliderLiveFeedback + kControlSliderHasTickMarks + kControlSliderReverseDirection
Slider with a rectangular, non-directional indicator.	8	56 kControlSliderProc + kControlSliderNonDirectional
Slider with live feedback and a rectangular, non-directional indicator.	9	57 kControlSliderProc + kControlSliderLiveFeedback + kControlSliderNonDirectional

Control Values

Control Value	*Content*
Initial	Appropriate value between -32768 and 32768. For the tick mark variant, the number of ticks required. Reset to the minimum setting after creation.
Minimum	-32768 to 32768.
Maximum	-32768 to 32768. When the maximum setting is equal to the minimum setting, the slider is inactive.

Control Part Codes

Constant	*Value*	*Description*
kControlIndicatorPart	129	Event occurred in the indicator

Programmatic Creation

Slider controls may be created programmatically using the function `CreateSliderControl`:

```
OSStatus  CreateSliderControl(WindowRef window,const Rect *boundsRect,SInt32 value,
                              SInt32 minimum,SInt32 maximum,
                              ControlSliderOrientation orientation,UInt16 numTickMarks,
                              Boolean liveTracking,ControlActionUPP liveTrackingProc,
                              ControlRef *outControl);
```

Relevant constants are

Parameter	*Constants*
orientation	kControlSliderPointsDownOrRight kControlSliderPointsUpOrLeft kControlSliderDoesNotPoint

Group Boxes-Embedding Control

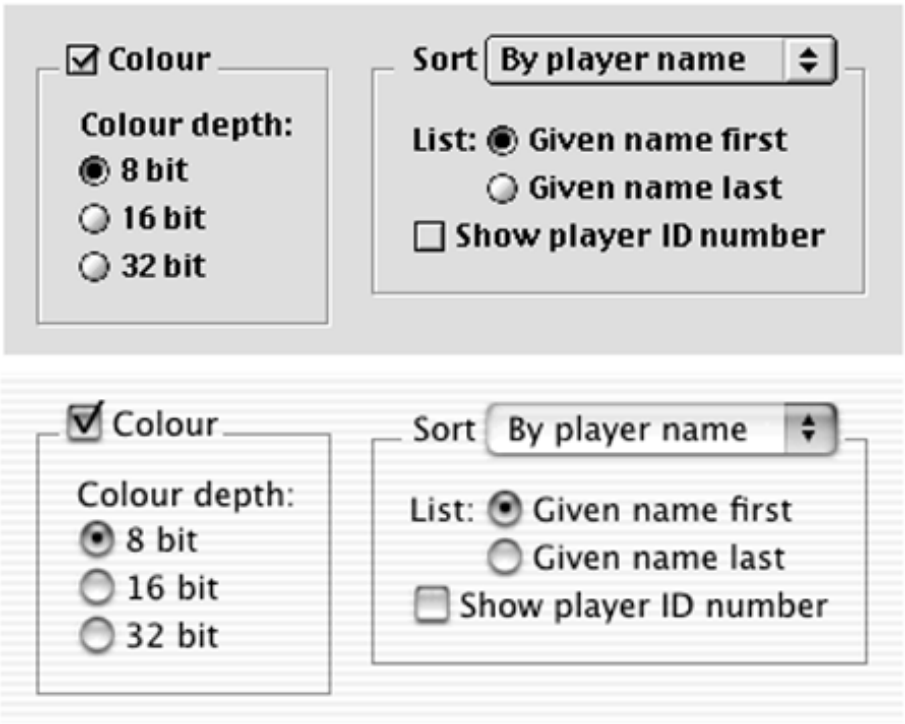
Group boxes are embedding controls used to group a number of related items. They may be either **primary** or **secondary.**

A group box can be untitled or it can have a text title, a pop-up menu title, or a checkbox title. Group boxes with a pop-up menu title are useful for displaying a variety of related settings in a limited space. Group boxes with a checkbox title are useful for indicating that a group of settings may be deactivated by the user.

Secondary group boxes are generally used for grouping subsidiary information.

Typical group boxes are shown in Fig 7.

Figure 7. TYPICAL GROUP BOXES



Variants and Control Definition IDs

The group box CDEF resource ID is 10. The six available variants and their control definition IDs are as follows:

Variant	*Var Code*	*Control Definition ID*
Primary group box, text title.	0	160 kControlGroupBoxTextTitleProc
Primary group box, checkbox title.	1	161 kControlGroupBoxCheckBoxProc
Primary group box, pop-up button title.	2	162 kControlGroupBoxPopupButtonProc
Secondary group box, text title.	4	164 kControlGroupBoxSecondaryTextTitleProc
Secondary group box, checkbox title.	5	165 kControlGroupBoxSecondaryCheckBoxProc
Secondary group box, pop-up button title.	6	166 kControlGroupBoxSecondaryPopupButtonProc

Control Values

Control Value	*Content*
Initial	Ignored if group box has text title If the group box has a checkbox or pop-up button title, same value as the checkbox or pop-up button
Minimum	Ignored if group box has text title. If the group box has a checkbox or pop-up button title, same minimum setting as the checkbox or pop-up button
Maximum	Ignored if group box has text title. If the group box has a checkbox or pop-up button title, same maximum setting as the checkbox or pop-up button

Control Data Tag Constant

Control Data Tag Constant	*Meaning and Data Type Returned or Set*
`kControlGroupBoxMenuRefTag`	Gets the menu reference of a group box. **Data type returned or set:** `MenuRef`
`kControlGroupBoxFontStyleTag`	Gets or sets the font style. **Data type returned or set:** `ControlFontStyleRec`
`kControlGroupBoxTitleRectTag`	Gets the rectangle containing the group box title (and any associated control, such as a checkbox). **Data type returned:** `Rect`

Control Part Codes

Constant	*Value*	*Description*
`kControlNoPart`	`0`	Returned if the group box title is a text title. If the group box title is a checkbox title or a pop-up menu button title, the user tracked completely out of the control.
`kControlButtonPart`	`10`	The group box title is a checkbox title and the check box was hit. The group box title is a pop-up menu button and the mouse button was released over the button.
`kControlMenuPart`	`2`	The group box title is a pop-up menu button and the mouse button was released in the menu.

Programmatic Creation

Group box controls may be created programmatically using the functions `CreateGroupBoxControl`, `CreateCheckGroupBoxControl`, and `CreatePopupGroupBoxControl`:

```
OSStatus  CreateGroupBoxControl(WindowRef window,const Rect *boundsRect,
                                CFStringRef title,Boolean primary,
                                ControlRef *outControl);

OSStatus  CreateCheckGroupBoxControl(WindowRef window,const Rect *boundsRect,
                                     CFStringRef title,SInt32 initialValue,
                                     Boolean primary,Boolean autoToggle,
                                     ControlRef *outControl);

OSStatus  CreatePopupGroupBoxControl(WindowRef window,const Rect *boundsRect,
                                     CFStringRef title,Boolean primary,SInt16 menuID,
                                     Boolean variableWidth,SInt16 titleWidth,
                                     SInt16 titleJustification,Style titleStyle,
                                     ControlRef *outControl);
```

Clock

Clock controls are a combination of an edit text control and little arrows that display date or time. The displayed date and time may

be changed using the little arrows or by typing in the edit text control. Fig 8 shows a clock control displaying a date.

Figure 8. CLOCK CONTROL DISPLAYING DATE



The clock control supports keyboard navigation and keyboard focus. If the control is made inactive, the user will be unable to change the displayed date or time values; however the correct date and time will continue to be displayed.

Variants and Control Definition IDs

The clock control CDEF resource ID is 15. The four available variants and their control definition ID are follows:

Variant	*Var Code*	*Control Definition ID*
Clock control displaying hour/ minutes.	`0`	`240` `kControlClockTimeProc`
Clock control displaying hours/ minutes/seconds.	`1`	`241` `kControlClockTimeSecondsProc`
Clock control displaying date/ month/year.	`2`	`242` `kControlClockDateProc`
Clock control displaying month/ year.	`3`	`243` `kControlClockMonthYearProc`

Control Values

Control Value	*Content*
Initial	One or more of the clock value flags. (See "Clock Value Flag Constants.") Reset to 0 after creation.
Minimum	Reserved. Set to 0.
Maximum	Reserved. Set to 0.

Clock Value Flag Constants

Constant	*Value*	*Description*
`kControlClockNoFlags`	`0`	Indicates that clock is editable but does not display the current "live" time.
`kControlClockIsDisplayOnly`	`1`	When only this bit is set, the clock is not editable. When this bit and the `kControlClockIsLive` bit is set, the clock automatically updates on idle (clock will have the current time).
`kControlClockIsLive`	`2`	When only this bit is set, the clock automatically updates on idle and any changes to the clock affect the system clock. When this bit and the `kControlClockIsDisplayOnly` bit is set, the clock automatically updates on idle (clock will have the current time), but is not editable.

Control Data Tag Constant

Control Data Tag Constant	*Meaning and Data Type Returned or Set*
`kControlClockLongDateTag`	Gets or sets the clock control's time or date. **Data type returned or set:** `LongDateRec` structure.

`kControlClockFontStyleTag`	Gets or sets the font style. **Data type returned or set:** `ControlFontStyleRec`
`kControlClockAnimatingTag`	Starts and stops clock animation. Use only if a call to `GetControlFeatures` reveals that the control's `kControlIdlesWithTimer` flag is set. **Data type set:** `Boolean`

Control Part Codes

Constant	*Value*	*Description*
`kControlClockPart`	`8`	Event occurred in a clock control.
`kControlClockHourDayPart`	`9`	Event occurred in the part that contains the hour.
`kControlClockMinuteMonthPart`	`10`	Event occurred in the part that contains the minute or the month.
`kControlClockSecondYearPart`	`11`	Event occurred in the part that contains the second or the year.
`kControlClockAMPMPart`	`12`	Event occurred in the part that contains the AM/PM information.

Programmatic Creation

Clock controls may be created programmatically using the function `CreateClockControl`:

```
OSStatus  CreateClockControl(WindowRef window,const Rect *boundsRect,
                             ControlClockType clockType,ControlClockFlags clockFlags,
                             ControlRef *outControl);
```

Relevant constants are

Parameter	*Constants*

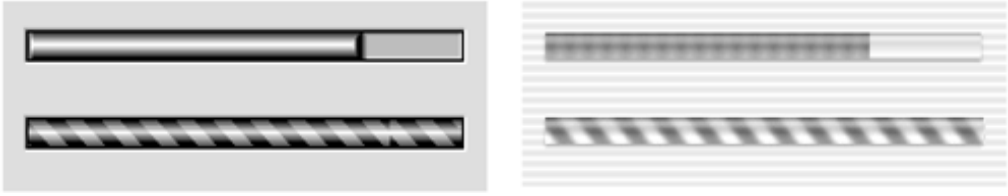
`clockType`	`kControlClockTypeHourMinute`
	`kControlClockTypeHourMinuteSecond`
	`kControlClockTypeMonthDayYear`
	`kControlClockTypeMonthYear`

`clockFlags`	See "Clock Value Flag Constants."

Progress and Relevance Bars

Progress bars are used to indicate capacity or the current status of a lengthy operation. Two types of Progress bars can be used: an **indeterminate progress bar,** which shows that an operation is occurring but does not indicate its current status; a **determinate progress bar,** which shows how much of the operation has been completed. The two types are shown in Fig 9.

Figure 9. PROGRESS INDICATORS



On Mac OS 8/9, the progress bar control is often used as a **relevance bar.** On Mac OS X, a separate relevance bar control, which can only be created programmatically, is available.

Variants and Control Definition IDs

The progress bar CDEF resource ID is 5. The two available variants and their control definition IDs are as follows:

Variant	*Var Code*	*Control Definition ID*
Progress bar. To make the control determinate or indeterminate, set the `kControlProgressBarIndeterminateTag` constant. (See "Control Data Tag Constant Relevant to Progress bars.")	`0`	80 `kControlProgressBarProc`
Relevance bar.	1	81 `kControlRelevanceBarProc`

Progress Bar Control Values

Control Value	*Content*
Initial	Appropriate value between -32768 and 32767
Minimum	-32768 to 32767
Maximum	-32768 to 32767

Control Data Tag Constant

Control Data Tag Constant	*Meaning and Data Type Returned or Set*
`kControlProgressBarIndeterminateTag`	Gets or sets whether a progress bar is determinate or indeterminate. **Data type returned or set:** `Boolean`. If `true`, switches to an indeterminate progress bar. If `false`, switches to an determinate progress bar.
`kControlProgressBarAnimatingTag`	Starts and stops progress bar animation. Use only if a call to `GetControlFeature`s reveals that the control's `kControlIdlesWithTimer` flag is set. **Data type returned or set:** `Boolean`

Programmatic Creation

Progress bars may be created programmatically using the function `CreateProgessBarControl`:

```
OSStatus  CreateProgressBarControl(WindowRef window,const Rect *boundsRect,SInt32 value,
                                   SInt32 minimum,SInt32 maximum,Boolean indeterminate,
                                   ControlRef *outControl);
```

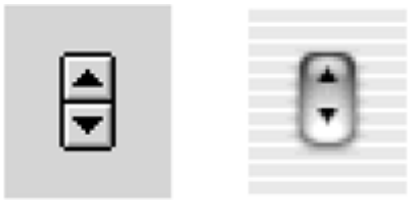
On Mac OS X only, relevance bars may be created programmatically using the function `CreateRelevanceBarControl`:

```
OSStatus  CreateRelevanceBarControl(WindowRef window,const Rect *boundsRect,SInt32 value,
                                    SInt32 minimum,SInt32 maximum,
                                    ControlRef *outControl);
```

Little Arrows

The little arrows control which consists of two opposing arrows, provide a means of increasing or decreasing values (see Fig 10). They should always be accompanied by a label that identifies the content to which the control relates.

Figure 10. LITTLE ARROWS



The displayed value is incremented or decremented by one unit of change when the user clicks the up or down arrow. If the user clicks an arrow and holds the mouse button down, the value increases or decreases until the user releases the button. The unit of change should depend on the content.

Variant and Control Definition ID

The little arrows CDEF resource ID is 6. The one available variant and its control definition ID is as follows:

Variant	*Var Code*	*Control Definition ID*
Little arrows	`0`	`96` `kControlLittleArrowsProc`

Control Values

Control Value	*Content*
Initial	Appropriate value between -32768 and 32767
Minimum	-32768 and 32767
Maximum	-32768 and 32767

Control Part Codes

Constant	*Value*	*Description*
`kControlUpButtonPart`	`20`	Event occurred in up arrow
`kControlDownButtonPart`	`21`	Event occurred in down arrow

Programmatic Creation

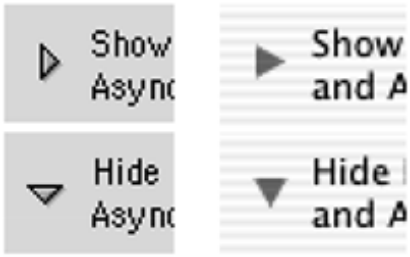
Little arrows controls may be created programmatically using the function `CreateLittleArrowsControl`:

```
OSStatus  CreateLittleArrowsControl(WindowRef window,const Rect *boundsRect,SInt32 value,
                                    SInt32 minimum,SInt32 maximum,SInt32 increment,
                                    ControlRef *outControl);
```

Disclosure Triangles

Disclosure triangles are used to display and hide additional information in a window and also to reveal the contents of folders in list views (see Fig 11). They have two possible values: 0 for collapsed and 1 for expanded. The first click on the control rotates the triangle downwards. The second click rotates the triangle back to the original orientation.

Figure 11. DISCLOSURE TRIANGLES



Variants and Control Definition IDs

The disclosure triangle CDEF resource ID is 4. The four available variants and their control definition ID is as follows:

Variant	*Var Code*	*Control Definition ID*
Disclosure triangle.	`0`	64 `kControlTriangleProc`
Left-facing disclosure triangle.	`1`	65 `kControlTriangleLeftFacingProc`
Auto-tracking disclosure triangle. This variant maintains its last value, so it knows what transition is taking place when a `SetControlValue` is called on it (expanded to collapsed, or vice versa). (See "Control Data Tag Constants.")	`2`	66 `kControlTriangleAutoToggleProc`
Left-facing, auto-tracking disclosure triangle.	`3`	67 `kControlTriangleLeftFacingAutoToggleProc`

Control Values

Control Value	*Content*
Initial	0 (collapsed) or 1 (expanded)
Minimum	0 (collapsed)
Maximum	1 (expanded)

Control Data Tag Constant

Control Data Tag Constant	*Meaning and Data Type Returned or Set*
`kControlTriangleLastValueTag`	Gets or sets the last value of a disclosure triangle. Used primarily for setting up a disclosure triangle properly when using the `kControlTriangleAutoToggleProc` variant. **Data type returned or set:** `SInt16`

Helper Function

The helper function `SetDisclosureTriangleLastValue` may be used in lieu of `SetControlData` with the `kControlTriangleLastValueTag` control data tag.

Control Part Codes

Constant	*Value*	*Description*
`kControlTrianglePart`	`4`	Event occurred in a disclosure triangle

Programmatic Creation

Disclosure triangle controls may be created programmatically using the function `CreateDisclosureTriangleControl`:

```
OSStatus  CreateDisclosureTriangleControl(WindowRef window,const Rect *boundsRect,
                                          ControlDisclosureTriangleOrientation orientation,
                                          CFStringRef title,SInt32 initialValue,
                                          Boolean drawTitle,Boolean autoToggles,
                                          ControlRef *outControl);
```

Relevant constants are

Parameter	*Constants*

`orientation`	`kControlDisclosureTrianglePointDefault`
	`kControlDisclosureTrianglePointRight`
	`kControlDisclosureTrianglePointLeft`

Picture

Picture controls display pictures.

Variants and Control Definition IDs

The picture control CDEF resource ID is 19. The two available variants and their control definition IDs are as follows:

Variant	*Var Code*	*Control Definition ID*
Picture control.	`0`	`304` `kControlPictureProc`
Non-tracking picture control. If hit, immediately returns the control part code `kControlNoPart` and does not track.	`1`	`305` `kControlPictureNoTrackProc`

Control Values

Control Value	*Content*
Initial	Resource ID of the `'PICT'` resource you wish to display. Reset to 0 after creation.
Minimum	Reserved. Set to 0.
Maximum	Reserved. Set to 0.

Control Data Tag Constant

Control Data Tag Constant	*Meaning and Data Type Returned or Set*
`kControlPictureHandleTag`	Sets picture **Data type set:** `PicHandle`

Control Part Codes

Constant	*Value*	*_Description*
`kControlPicturePart`	`6`	Event occurred in a picture control

Programmatic Creation

Picture controls may be created programmatically using the function `CreatePictureControl`:

```
OSStatus  CreatePictureControl(WindowRef window,const Rect *boundsRect,
                               const ControlButtonContentInfo *content,Boolean dontTrack,
                               ControlRef *outControl);
```

Icon

Icon controls display color icons and icons from an icon suite.

A non-tracking variant is available for use in dialogs that have an embedding hierarchy and want an icon. This variant just returns the part hit immediately; it does not actually track the mouse.

Variants and Control Definition IDs

The icon CDEF resource ID is 20. The four available variants and their control definition IDs are as follows:

Variant	*Var Code*	*Control Definition ID*
Icon control.	`0`	`320` `kControlIconProc`
Non-tracking icon. If hit, immediately returns the control part code `kControlIconPart` without tracking.	`1`	`321` `kControlIconNoTrackProc`
Icon suite.	`2`	`322` `kControlIconSuiteProc`
Non-tracking icon suite. If hit, immediately returns the control part code `kControlIconPart` without tracking.	`3`	`323` `kControlIconSuiteNoTrackProc`
Supports all standard types of icon-based content.	`4`	`324` `kControlIconRefProc`
Supports all standard types of icon-based content. Non-tracking variant.	`5`	`325` `kControlIconRefNoTrackProc`

Control Values

Control Value	*Content*
Initial	Resource ID of the `'cicn'`, `'ICON'`, or icon suite resource you wish to display. (The icon suite variant looks for an icon suite. If not found, it looks for a `'cicn'` or `'ICON'` resource.) Reset to 0 after creation.
Minimum	Reserved. Set to 0.
Maximum	Reserved. Set to 0.

Control Data Tag Constants

Control Data Tag Constant	*Meaning and Data Type Returned or Set*
`kControlIconTransformTag`	Gets or sets a transform that is added to the standard transform for an icon. (See Chapter 13.) **Data type returned or set:** `IconTransformType`

`kControlIconAlignmentTag`	Gets or sets an icon's position. (See Chapter 13.) **Data type returned or set:** `IconAlignmentType`
`kControlIconResourceIDTag`	Gets or sets the resource ID of the icon to use. **Data type returned or set:** `Sint16`
`kControlIconContentTag`	Gets or sets the type of content to be used in an icon control. **Data type returned or set:** `ControlButtonContentInfo`

Control Part Codes

Constant	*Value*	*Description*
`kControlIconPart`	`7`	Event occurred in an icon control

Programmatic Creation

Icon controls may be created programmatically using the function `CreateIconControl`:

```
OSStatus  CreateIconControl(WindowRef window,const Rect *boundsRect,
                            const ControlButtonContentInfo *icon,Boolean dontTrack,
                            ControlRef *outControl);
```

Window Headers-Embedding Control

Window headers are rectangular embedding controls which should be located at the top of a window's content region and used to provide information about the window's contents.

The list view header variant is similar to the main variant, but removes the line that separates a standard window header from the content area.

Variants and Control Definition IDs

The window header CDEF resource ID is 21. The two available variants and their control definition IDs are as follows:

Variant	*Var Code*	*Control Definition ID*
Window header	`0`	`336` `kControlWindowHeaderProc`
Window list view header	1	337 `kControlWindowListViewHeaderProc`

Control Values

Control Value	*Content*
Initial	Reserved. Set to 0.
Minimum	Reserved. Set to 0.
Maximum	Reserved. Set to 0.

Programmatic Creation

Window header controls may be created programmatically using the function CreateWindowHeaderControl:

```
OSStatus  CreateWindowHeaderControl(WindowRef window,const Rect *boundsRect,
                                    Boolean isListHeader,ControlRef *outControl);
```

Placard-Embedding Control

Placards are rectangular embedding controls used to display information. They can also be used simply as background fill for a control area. Typically, placards are used as a small information panel placed at the bottom of a window.

Variants and Control Definition IDs

The placard CDEF resource ID is 14. The one available variant and its control definition ID is as follows:

Variant	*Var Code*	*Control Definition ID*
Placard	0	224 kControlPlacardProc

Future versions will provide a push button variant.

Control Values

Control Value	*Content*
Initial	Reserved. Set to 0.
Minimum	Reserved. Set to 0.
Maximum	Reserved. Set to 0.

Programmatic Creation

Placard controls may be created programmatically using the function CreatePlacardControl:

```
OSStatus  CreatePlacardControl(WindowRef window,const Rect *boundsRect,
                               ControlRef *outControl);
```

Static Text

Static text controls display static text, that is, text that cannot be changed by the user.

Variant and Control Definition ID

The static text control CDEF resource ID is 18. The one available variant and its control definition ID is as follows:

Variant	*Var Code*	*Control Definition ID*
Static text control	0	288 kControlStaticTextProc

Control Values

Control Value	*Content*
Initial	Reserved. Set to 0.
Minimum	Reserved. Set to 0.
Maximum	Reserved. Set to 0.

Control Data Tag Constants

Control Data Tag Constant	*Meaning and Data Type Returned or Set*
`kControlStaticTextTextTag`	Gets or sets text. **Data type returned or set:** character buffer
`kControlStaticTextTextHeightTag`	Gets the height of text. **Data type returned or set:** `SInt16`
`kControlStaticTextStyleTag`	Gets or sets the font style. **Data type returned or set:** `ControlFontStyleRec`
`kControlStaticTextTruncTag`	Gets or sets how text is truncated at end of a line. **Data type returned or set:** `TruncCode`. The value `truncEnd` indicates that characters are truncated off the end of the string. `truncMiddle` indicates that characters are truncated from the middle of the string. The default is `-1`, which indicates that no truncation occurs and, instead, the text is wrapped.
`kControlStaticTextCFStringTag`	Gets or sets a CFString reference. **Data type returned or set:** `CFStringRef`

Programmatic Creation

Static text controls may be created programmatically using the function

`CreateStaticTextControl`:

```
OSStatus  CreateStaticTextControl(WindowRef window,const Rect *boundsRect,
                                  CFStringRef text,const ControlFontStyleRec *style,
                                  ControlRef * outControl);
```

Separator Lines

Separator lines are vertical or horizontal lines used to visually separate groups of controls. The orientation of the bounding rectangle determines the orientation of the line.

Variants and Control Definition IDs

The separator line CDEF resource ID is 9. The one available variant and its control definition ID is as follows:

Variant	*Var Code*	*Control Definition ID*
Separator line	`0`	`144` `kControlSeparatorLineProc`

Control Values

Control Value	*Content*

Initial	Reserved. Set to 0.
Minimum	Reserved. Set to 0.
Maximum	Reserved. Set to 0.

Programmatic Creation

Separator line controls may be created programmatically using the function `CreateSeparatorControl`:

```
OSStatus  CreateSeparatorControl(WindowRef window,const Rect *boundsRect,
                                 ControlRef *outControl);
```

Pop-up Arrows

The pop-up arrow control simply draws the pop-up glyph.

Variants and Control Definition IDs

The pop-up arrow CDEF resource ID is 12. The eight available variants and their control definition IDs are as follows:

Variant	*Var Code*	*Control Definition ID*
Large, east-facing pop-up arrow	`0`	`192` `kControlPopupArrowEastProc`
Large, west-facing pop-up arrow	`1`	`193` `kControlPopupArrowWestProc`
Large, north-facing pop-up arrow	`2`	`194` `kControlPopupArrowNorthProc`
Large, south-facing pop-up arrow	`3`	`195` `kControlPopupArrowSouthProc`
Small, east-facing pop-up arrow	`4`	`196` `kControlPopupArrowSmallEastProc`
Small, west-facing pop-up arrow	`5`	`197` `kControlPopupArrowSmallWestProc`
Small, north-facing pop-up arrow	`6`	`198` `kControlPopupArrowSmallNorthProc`
Small, south-facing pop-up arrow	`7`	`199` `kControlPopupArrowSmallSouthProc`

Control Values

Control Value	*Content*
Initial	Reserved. Set to 0.
Minimum	Reserved. Set to 0.
Maximum	Reserved. Set to 0.

Programmatic Creation

Pop-up arrow controls may be created programmatically using the function `CreatePopupArrowControl`:

```
OSStatus  CreatePopupArrowControl(WindowRef window,const Rect *boundsRect,
                                  ControlPopupArrowOrientation orientation,
                                  ControlPopupArrowSize size,ControlRef *outControl);
```

Relevant constants are

Parameter	*Constants*

orientation	kControlPopupArrowOrientationEast kControlPopupArrowOrientationWest kControlPopupArrowOrientationNorth kControlPopupArrowOrientationSouth
size	kControlPopupArrowSizeNormal kControlPopupArrowSizeSmall

Radio Groups-Embedding Control

Radio groups are embedding controls which relieve your application of much of the work involved in managing a group of radio buttons (or bevel buttons that are intended to operate like radio buttons). For example, if a group of radio buttons are embedded in a radio group control, the radio group control handles the checking and unchecking of the radio buttons when the user clicks on one of them. The current value of the radio group control represents the radio button currently selected.

Variant and Control Definition ID

The radio group CDEF resource ID is 26. The one available variant and its control definition ID is as follows:

Variant	*Var Code*	*Control Definition ID*
Radio group	0	416 kControlRadioGroupProc

Control Values

Control Value	*Content*
Initial	Set to 0 on creation. Reset to the index of the currently selected embedded radio control after creation. If the currently selected control does not support radio behavior, this value will be set to 0 and the control will be deselected. To deselect all controls, set to 0.
Minimum	Set to 0.
Maximum	Set to 0 on creation. Reset to the number of embedded controls as controls are added.

Control Part Codes

Constant	*Value*	*Description*
kControlRadioGroupPart	27	Event occurred in a radio group

Programmatic Creation

Radio group controls may be created programmatically using the function CreateRadioGroupControl:

```
OSStatus  CreateRadioGroupControl(WindowRef window,const Rect *boundsRect,
                                  ControlRef *outControl);
```

Chasing Arrows

Chasing arrows are a simple animation used to indicate that an asynchronous background process is occurring, in other words a process which does not display a dialog containing a progress bar (see Fig 12).

Figure 12. FRAMES OF A CHASING ARROWS ANIMATION



Variant and Control Definition ID

The chasing arrows CDEF resource ID is 7. The one available variant and its control definition ID is as follows:

Variant	*Var Code*	*Control Definition ID*
Chasing arrows	`0`	`112` `kControlChasingArrowsProc`

Control Values

Control Value	*Content*
Initial	Reserved. Set to 0.
Minimum	Reserved. Set to 0.
Maximum	Reserved. Set to 0.

Control Data Tag Constant

Control Data Tag Constant	*Meaning and Data Type Returned or Set*
`kControlChasingArrowsAnimatingTag`	Starts and stops chasing arrows animation. Use only if a call to `GetControlFeatures` reveals that the control's `kControlIdlesWithTimer` flag is set. **Data type returned or set:** `Boolean`.

Programmatic Creation

Asynchronous arrow controls may be created programmatically using the function `CreateChasingArrowsControl`:

```
OSStatus  CreateChasingArrowsControl(WindowRef window,const Rect *boundsRect,
                                     ControlRef *outControl);
```

User Panes-Embedding Control

The user pane has two main uses:

- It can be used as an embedding control, that is, other controls may be embedded within it. It is thus particularly useful for grouping together the controls belonging to an individual pane of a tab control or pop-up button group box.
- It provides a way to hook in application-defined (callback) functions, known as **user pane functions,** which perform actions such as drawing, hit testing, tracking, and so on.

Variants and Control Definition IDs

The user pane CDEF resource ID is 16. The one available variant and its control definition ID is as follows:

Variant	*Var Code*	*Control Definition ID*
User pane	`0`	`256` `kControlUserPaneProc`

Control Values

Control Value	*Content*
Initial	One or more of the control feature constants. (See "Defining Your Own User Pane Functions.") Reset to 0 after creation.
Minimum	Ignored. After user pane creation, reset to a setting between -32768 to 32767.
Maximum	Ignored. After user pane creation, reset to a setting between -32768 to 32767.

Control Data Tag Constants

The control data tag constants relevant to user panes all relate to user pane functions. See Defining Your Own User Pane Function.

Programmatic Creation

User Pane controls may be created programmatically using the function `CreateUserPaneControl`:

```
OSStatus  CreateUserPaneControl(WindowRef window,const Rect *boundsRect,
                                UInt32 features,ControlRef *outControl);
```

Scrolling Text Box

The scrolling text box implements a scrolling box of non-editable text. There are two variants:

- The standard variant, which has a scroll bar.
- The auto-scrolling variant, which does not have a scroll bar. This variant needs two pieces of information to work: the delay (in ticks) before the scrolling starts; the time (in ticks) between scrolls. By default, the text will scroll one pixel at a time, although this may be changed via `SetControlData`.

Variants and Control Definition IDs

The scrolling text box CDEF resource ID is 27. The two available variants and their control definition IDs are as follows:

Variant	*Var Code*	*Control Definition ID*	
Standard	`0`	`432`	`kControlScrollTextBoxProc`
Auto-scrolling	`1`	`433`	`kControlScrollTextBoxAutoScrollProc`

Control Values

Control Value	*Content*
Initial	Resource ID of a `'TEXT'` and, optionally, a `'styl'` resource.
Minimum	For the standard variant, set to 0. For the auto-scrolling variant, the delay (in ticks) before scrolling begins. (This delay is also used between when the scrolling completes and when it begins again.)
Maximum	For the standard variant, set to 0. For the auto-scrolling variant, the delay (in ticks) between each unit of scrolling.

Control Data Tag Constants

Control Data Tag Constant	*Meaning and Data Type Returned or Set*
`KControlScrollTextBoxDelayBefore AutoScrollTag`	Gets or sets the number of ticks of delay before the initial scrolling of an auto-scrolling text box begins. **Data type returned or set:** `UInt32`
`KControlScrollTextBoxDelay BetweenAutoScrollTag`	Gets or sets the number of ticks of delay between each unit of scrolling in an auto-scrolling text box. **Data type returned or set:** `UInt32`
`KControlScrollTextBoxAutoScroll AmountTag`	Gets or sets the number of pixels by which an auto scrolling text box scrolls. (The default is 1.) **Data type returned or set:** `UInt16`
`kControlScrollTextBoxContentsTag`	Sets the ID of a `'TEXT'` resource and, optionally, a `'styl'` resource, to be used as the content of a standard or auto-scrolling text box. **Data set:** `Sint16`

Programmatic Creation

Scrolling text box controls may be created programmatically using the function `CreateScrollingTextBoxControl`:

```
OSStatus  CreateScrollingTextBoxControl(WindowRef window,const Rect *boundsRect,
                                        SInt16 contentResID,Boolean autoScroll,
                                        UInt32 delayBeforeAutoScroll,
                                        UInt32 delayBetweenAutoScroll,
                                        UInt16 autoScrollAmount,ControlRef *outControl);
```

Disclosure Button

The disclosure button is available only on Mac OS X, and can only be created programmatically using the function `CreateDisclosureButtonControl` (see Fig 13):

```
OSStatus  CreateDisclosureButtonControl(WindowRef inWindow,const Rect *inBoundsRect,
                                        SInt32 inValue,Boolean inAutoToggles,
                                        ControlRef *  outControl);
```

Figure 13. DISCLOSURE BUTTON



The height of the button is fixed at 20 pixels. The button will be as wide as the specified rectangle and will be centered in that rectangle vertically.

Relevant constants are:

Parameter	*Constants*

`inValue`	`kControlDisclosureButtonClosed` `kControlDisclosureButtonDisclosed`

Edit Unicode Text

The edit Unicode text control is available only on Mac OS X, and can only be created programmatically using the function `CreateEditUnicodeTextControl`:

```
OSStatus  CreateEditUnicodeTextControl(WindowRef window,const Rect *boundsRect,
                                       CFStringRef text,Boolean isPassword,
                                       const ControlFontStyleRec *style,
                                       ControlRef *outControl);
```

The control data tags applicable to the edit text control also apply to the edit Unicode text control, except that the `kControlEditTextTEHandleTag` tag must not be used.

Round Button

The round button is available only on Mac OS X, and can only be created programmatically using the function `CreateRoundButtonControl`:

```
OSStatus  CreateRoundButtonControl(WindowRef nWindow,const Rect *inBoundsRect,
                                   ControlRoundButtonSize inSize,
                                   ControlButtonContentInfo *inContent,
                                   ControlRef *outControl);
```

Relevant constants are

Parameter	*Constants*
`inSize`	`kControlRoundButtonNormalSize` (20-pixel diameter)
	`kControlRoundButtonLargeSize` (25-pixel diameter)

Control Data Tag Constants

Control Data Tag Constant	*Meaning and Data Type Returned or Set*
`kControlRoundButtonContentTag`	Sets button content **Data type set:** `ControlButtonContentInfo`
`kControlRoundButtonSizeTag`	Sets button size **Data type set:** `ControlRoundButtonSize`

Small Versions of Controls

Human Interface Guidelines permit the use of small versions of certain controls, which should only be used when space is at a premium. Full size and small controls should not be mixed in the same window.

The following lists those controls described in this chapter that are available in small versions, and describes how to create them.

Control	*Mac OS X*	*Mac OS 8/9*
Tab	Pass `kControlTabSizeSmall` in `CreateTabsControl` call or use a small variant if creating from a `'CNTL'` resource. Alternatively, if the control is created large, call `SetControlData` with the `kControlSizeTag` tag.	Pass `kControlTabSizeSmall` in `CreateTabsControl` call or use a small variant if creating from a `'CNTL'` resource.
Edit text	Make the control's rectangle 13 pixels high and call `SetControlFontStyle` to set the control's font to the small system font.	Make the control's rectangle 13 pixels high and call `SetControlFontStyle` to set the control's font to the small system font.
Slider	Call `SetControlData` with the `kControlSizeTag` tag.	(Not available.)

Pop-up arrow	Pass `kControlPopupArrowSizeSmall` in `CreatePopupArrowControl`call or use a small variant if creating from a `'CNTL'` resource. Alternatively, if the control is created large, call `SetControlData` with the `kControlSizeTag` to make it small.	Pass `kControlPopupArrowSizeSmall` in `CreatePopupArrow Control` call or use a small variant if creating from a `'CNTL'` resource.
Round button	Call `SetControlData` with the `kControlSizeTag` tag.	(Not available.)

Idle Processing

The following four controls need to perform idle processing for the reasons indicated:

Control	*Reason For Idle Processing*
Progress bar (indeterminate variant)	Animate the indicator
Clocks (when the `kControlClockIsLive` clock value flag constant is used)	Update the clock
Chasing arrows	Animate the arrows
Edit text controls	Cause TextEdit to be called to blink the insertion point caret
Scrolling text box (auto-scrolling variant)	Scroll the text automatically

When these controls are displayed in a document window, your application may need to call `IdleControls` at the appropriate interval in your event loop, otherwise the animations, and so on, will not occur. `IdleControls` sends a particular message to the CDEF, which responds appropriately.

The call to `IdleControls` is not necessary on Mac OS X because Mac OS X controls idle themselves automatically using their own internal timers. However, as of the time of writing, the call is necessary on Mac OS 8/9 because controls on Mac OS 8/9 do not yet have internal timers.

When these controls are displayed in a dialog on Mac OS 8/9, `ModalDialog` calls `IdleControls` for you.

You can determine whether a control has an internal timer by calling `GetControlFeatures` and testing for the `kControlIdlesWithTimer` bit.

Defining Your Own Key Filter Function

You can attach a **key filter function** to edit text controls to filter key strokes or modify them on return. Your key filter function can change the keystroke, leave it alone, or block the CDEF from receiving it. For example, an edit text control could use a key filter function to allow only numeric values to be input in its field.

You would declare your key filter function as follows:

```
ControlKeyFilterResult  myKeyFilterFunction(ControlRef controlRef,SInt16* keyCode,
                                            SInt16 *charCode,EventModifiers *modifiers);
```

The Control Manager defines the data type `ControlKeyFilterUPP` to identify the universal procedure pointer for this function:

```
typedef STACK_UPP_TYPE(ControlKeyFilterProcPtr) ControlKeyFilterUPP;
```

As stated in the "Edit Text" section, the control data tag constant for getting and setting a key filter function is `kControlKeyFilterTag` and the data type returned or set is `ControlKeyFilterUPP`.

Example

The following example relates to an edit text control and assumes a key filter function named `numericFilter`.

```
#define kLeftArrow  0x1C
#define kRightArrow 0x1D
#define kUpArrow    0x1E
#define kDownArrow  0x1F
#define kBackspace  0x08
...

ControlKeyFilterUPP numericFilterUPP;
ControlRef          controlHdl;
...

// ................................................. get universal procedure pointer

numericFilterUPP = NewControlKeyFilterProc(numericFilter);

// ..................................... attach key filter function to the control

GetDialogItemAsControl(dialogPtr,itemNumber,&controlHdl);
SetControlData(controlHdl,kControlNoPart,kControlEditTextKeyFilterTag,
               sizeof(numericFilterUPP),&numericFilterUPP);

...

// ********************************************************************* numericFilter
//
// This function will be called each time the edit text control receives a keystroke.
// Keystrokes that are to be accepted must return kControlKeyFilterPassKey. Keystrokes
// that are to be blocked must return kControlKeyFilterBlockKey. This function blocks
// all but the numeric keys, the "dash" key, and the arrow keys.

ControlKeyFilterResult numericFilter(ControlRef control,SInt16 *keyCode,
                                     SInt16 *charCode,SInt16 *modifiers)
{
```

```
  if(((char) *charCode >= '0') && ((char) *charCode <= '9'))
    return kControlKeyFilterPassKey;

  switch(*charCode)
  {
    case '-':
    case kLeftArrow:
    case kRightArrow:
    case kUpArrow:
    case kDownArrow:
    case kBackspace:
      return kControlKeyFilterPassKey;
      break;

    default:
      SysBeep(10);
      return kControlKeyFilterBlockKey;
      break;
  }
}
```

Defining Your Own Edit Text Validation Function

A key filter function, however, does not cater for the case of pasting text to an edit text item. Accordingly, you will ordinarily want to combine an **edit text validation function** with the key filter function for a specific edit text control.

You would declare your edit text validation function as follows:

```
void  myEditTextValidationFunction(ControlRef controlHdl);
```

The following example ensures that, if a user supplied filename pasted to the edit text item contains one or more colons, those colons will be replaced with dashes.

```
ControlEditTextValidationUPP editTextValidatorUPP;
ControlRef                   controlHdl;
...

// ................................................ get universal procedure pointer

editTextValidatorUPP  = NewControlEditTextValidationProc(editTextValidator);

// ............................... attach edit text validation function to the control

GetDialogItemAsControl(dialogPtr,iEditText1,&controlHdl);
SetControlData(controlHdl,kControlNoPart,kControlEditTextValidationProcTag,
               sizeof(editTextValidatorUPP),&editTextValidatorUPP);
...

void  editTextValidator(ControlRef controlHdl)
{
  Str31 theText;
  Size  actualSize;
  UInt8 a;
  // ..................................... Get the text to be examined from the control

  GetControlData(controlHdl,kControlNoPart,kControlEditTextTextTag,sizeof(theText) -1,
                 (Ptr) &theText[1],&actualSize);

  // ..................................................... Set the length byte to the
  // .. number of characters in the text, limited to the current maximum for filenames

  if(actualSize <= 31)
    theText[0] = actualSize;
  else
    theText[0] = 31;

  // ................................................ Replace any colons with dashes

  for(a=1;a<=theText[0];a++)
  {
    if(theText[a] == ':')
      theText[a] = '-';
  }

  // ..................................................... You might want to add code
  // ... here to check whether any characters were replaced before bothering to redraw

  // ..................... Put the replaced text into the control and redraw the control
```

```
  SetControlData(controlHdl,kControlNoPart,kControlEditTextTextTag,theText[0],
                 (Ptr) &theText[1]);

  Draw1Control(controlHdl);
}
```

Defining Your Own User Pane Functions

As previously stated, one of the functions of a user pane is to provide a way to hook in application-defined (callback) functions that perform actions such as drawing, hit testing, tracking, and so on. Such application-defined (callback) functions are called **user pane functions.** User pane functions provide you with the ability to create a custom control without writing your own control definition function.

User Pane Functions

User pane functions are categorized as follows:

Function	***Description***
Draw	Draws the content of a user pane control.
Hit test	Returns the part code when a mouse-down event occurs.
Tracking	Tracks a control while the user holds down the mouse button. The function should track the control by repeatedly calling the action function specified in the `actionProc` parameter until the mouse button is released. When the mouse button is released, your function should return the part code of the control part that was tracked. This function will only be called if you have set the `kControlHandlesTracking` control feature bit on creation of the user pane control.
Idle	Performs idle processing. This function will only be called if you have set the `kControlWantsIdle` control feature bit on creation of the user pane control.
Key Down	Handles keyboard event processing. The function should handle the key pressed or released by the user and return the part code of the control where the keyboard event occurred. This function will only be called if you've set the `kControlSupportsFocus` control feature bit on creation of the user pane control.

Activate	Handles activate and deactivate event processing. The function should perform any special processing before the user pane becomes activated or deactivated. For example, it should deactivate its `TEHandle` or `ListHandle` if the user pane is about to be deactivated. This function will only be called if you have set the `kControlWantsActivate` control feature bit on creation of the user pane control.
Focus	Handle keyboard focus. The function is called in response to a change in keyboard focus. It should respond by changing keyboard focus based on the part code passed in the `action` parameter. This function will only be called if you have set the `kControlSupportsFocus` control feature bit on creation of the user pane control.
Background	Sets the background color or pattern (only for user panes that support embedding). The function should set the user pane background color or pattern to whatever is appropriate given the bit depth and device type passed in. This function will only be called if you have set the `kControlHasSpecialBackground` and `kControlSupportsEmbedding` control feature bits on creation of the user pane control.

Control Data Tag Constants and Universal Procedure Pointers

Once you have provided a user pane function, you call `SetControlData` with the control data tag constant representing the user pane function you want to set passed in the `inTagName` parameter and a universal procedure pointer to the user pane function passed in the `inData` parameter.

Control Data Tag Constants

The control data tag constants relating to user pane functions are as follows:

Control Data Tag Constant	*Meaning and Data Type Returned or Set*

`kControlUserPaneDrawProcTag`

Gets or sets a drawing function. Indicates that the Control Manager needs to draw a control.

Data type returned or set:
`ControlUserPaneDrawingUPP`

`kControlUserPaneHitTestProcTag`

Gets or sets a hit-testing function. Indicates that the Control Manager needs to determine if a control part was hit.

Data type returned or set:
`ControlUserPaneHitTestUPP`

`kControlUserPaneTrackingProcTag`

Gets or sets a tracking function. The tracking function will be called when a control definition function returns the `kControlHandlesTracking` feature bit in response to a `kControlMsgGetFeatures` message. Indicates that a user pane handles its own tracking.

Data type returned or set:
`ControlUserPaneTrackingUPP`

`kControlUserPaneIdleProcTag`

Gets or sets an idle function. The idle function will be called when a control definition function returns the `kControlWantsIdle` feature bit in response to `kControlMsgGetFeatures` message. Indicates that a user pane performs idle processing.

Data type returned or set:
`ControlUserPaneIdleUPP`

`kControlUserPaneKeyDownProcTag`

Gets or sets a key down function. The key down function will be called when a control definition function returns the `kControlSupportsFocus` feature bit in response to a `kControlMsgGetFeatures` message. Indicates that a user pane performs keyboard event processing.

Data type returned or set:
`ControlUserPaneKeyDownUPP`

`kControlUserPaneActivateProcTag`

Gets or sets an activate function. The activate function will be called when a control definition function returns the `kControlWantsActivate` feature bit in response to a `kControlMsgGetFeatures` message. Indicates that a user pane wants to be informed of activate and deactivate events.

Data type returned or set:
`ControlUserPaneActivateUPP`

`kControlUserPaneFocusProcTag`

Gets or sets a keyboard focus function. The keyboard focus function will be called when a control definition function returns the `kControlSupportsFocus` feature bit in response to a `kControlMsgGetFeatures` message. Indicates that a user pane handles keyboard focus.

Data type returned or set:
`ControlUserPaneFocusUPP`

`kControlUserPaneBackgroundProcTag`	Gets or sets a background function. The background function will be called when a control definition function returns the `kControlHasSpecialBackground` and `kControlSupportsEmbedding` feature bits in response to a `kControlMsgGetFeatures` message. Indicates that a user pane can set its background color or pattern. **Data type returned or set:** `ControlUserPaneBackgroundUPP`

Creating and Disposing of Universal Procedure Pointers

The following functions create and dispose of universal procedure pointers for user pane functions:

```
NewControlUserPaneDrawUPP         DisposeControlUserPaneDrawUPP
NewControlUserPaneHitTestUPP      DisposeControlUserPaneHitTestUPP
NewControlUserPaneTrackingUPP     DisposeControlUserPaneTrackingUPP
NewControlUserPaneIdleUPP         DisposeControlUserPaneIdleUPP
NewControlUserPaneKeyDownUPP      DisposeControlUserPaneKeyDownUPP
NewControlUserPaneActivateUPP     DisposeControlUserPaneActivateUPP
NewControlUserPaneFocusUPP        DisposeControlUserPaneFocusUPP
NewControlUserPaneBackgroundUPP   DisposeControlUserPaneBackgroundUPP
```

Main Constants, Data Types and Functions

Constants

Control Definition IDs

```
kControlBevelButtonSmallBevelProc              = 32
kControlBevelButtonNormalBevelProc             = 33
kControlBevelButtonLargeBevelProc              = 34
kControlSliderProc                             = 48
kControlSliderLiveFeedback                     = (1 << 0)
kControlSliderHasTickMarks                     = (1 << 1)
kControlSliderReverseDirection                 = (1 << 2)
kControlSliderNonDirectional                   = (1 << 3)
kControlTriangleProc                           = 64
kControlTriangleLeftFacingProc                 = 65
kControlTriangleAutoToggleProc                 = 66
kControlTriangleLeftFacingAutoToggleProc       = 67
kControlProgressBarProc                        = 80
kControlLittleArrowsProc                       = 96
kControlChasingArrowsProc                      = 112
kControlTabLargeNorthProc                      = 128
kControlTabSmallNorthProc                      = 129
kControlTabLargeSouthProc                      = 130
kControlTabSmallSouthProc                      = 131
kControlTabLargeEastProc                       = 132
kControlTabSmallEastProc                       = 133
kControlTabLargeWestProc                       = 134
kControlTabSmallWestProc                       = 135
kControlSeparatorLineProc                      = 144
kControlGroupBoxTextTitleProc                  = 160
kControlGroupBoxCheckBoxProc                   = 161
kControlGroupBoxPopupButtonProc                = 162
kControlGroupBoxSecondaryTextTitleProc         = 164
kControlGroupBoxSecondaryCheckBoxProc          = 165
kControlGroupBoxSecondaryPopupButtonProc       = 166
kControlImageWellProc                          = 176
kControlPopupArrowEastProc                     = 192
kControlPopupArrowWestProc                     = 193
kControlPopupArrowNorthProc                    = 194
kControlPopupArrowSouthProc                    = 195
kControlPopupArrowSmallEastProc                = 196
kControlPopupArrowSmallWestProc                = 197
kControlPopupArrowSmallNorthProc               = 198
kControlPopupArrowSmallSouthProc               = 199
kControlPlacardProc                            = 224
kControlClockTimeProc                          = 240
kControlClockTimeSecondsProc                   = 241
kControlClockDateProc                          = 242
kControlClockMonthYearProc                     = 243
kControlUserPaneProc                           = 256
kControlEditTextProc                           = 272
kControlEditTextDialogProc                     = 273
kControlEditTextPasswordProc                   = 274
kControlEditTextInlineInputProc                = 276
kControlStaticTextProc                         = 288
kControlPictureProc                            = 304
kControlPictureNoTrackProc                     = 305
kControlIconProc                               = 320
kControlIconNoTrackProc                        = 321
kControlIconSuiteProc                          = 322
kControlIconSuiteNoTrackProc                   = 323
```

```
kControlIconRefProc                     = 324
kControlIconRefNoTrackProc              = 325
kControlWindowHeaderProc                = 336
kControlWindowListViewHeaderProc        = 337
kControlListBoxProc                     = 352
kControlListBoxAutoSizeProc             = 353
kControlRadioGroupProc                  = 416
kControlScrollTextBoxProc               = 432
kControlScrollTextBoxAutoScrollProc     = 433
```

Control Variants

```
kControlNoVariant                       = 0
kControlUsesOwningWindowsFontVariant    = 1 << 3
```

Control Part Codes

```
kControlNoPart                          = 0
kControlLabelPart                       = 1
kControlMenuPart                        = 2
kControlTrianglePart                    = 4
kControlEditTextPart                    = 5
kControlPicturePart                     = 6
kControlIconPart                        = 7
kControlClockPart                       = 8
kControlClockHourDayPart                = 9
kControlClockMinuteMonthPart            = 10
kControlClockSecondYearPart             = 11
kControlClockAMPMPart                   = 12
kControlListBoxPart                     = 24
kControlListBoxDoubleClickPart          = 25
kControlImageWellPart                   = 26
kControlRadioGroupPart                  = 27
kControlButtonPart                      = 10
kControlCheckBoxPart                    = 11
kControlRadioButtonPart                 = 11
kControlUpButtonPart                    = 20
kControlDownButtonPart                  = 21
kControlPageUpPart                      = 22
kControlPageDownPart                    = 23
kControlIndicatorPart                   = 129
kControlDisabledPart                    = 254
kControlInactivePart                    = 255
```

Bevel Button Graphic Alignment

```
KControlBevelButtonAlignSysDirection    = -1
KControlBevelButtonAlignCenter          = 0
KControlBevelButtonAlignLeft            = 1
KControlBevelButtonAlignRight           = 2
KControlBevelButtonAlignTop             = 3
kControlBevelButtonAlignBottom          = 4
kControlBevelButtonAlignTopLeft         = 5
kControlBevelButtonAlignBottomLeft      = 6
kControlBevelButtonAlignTopRight        = 7
kControlBevelButtonAlignBottomRight     = 8
```

Bevel Button Text Alignment Values

```
KControlBevelButtonAlignTextSysDirection    = teFlushDefault
KControlBevelButtonAlignTextCenter          = teCenter
KControlBevelButtonAlignTextFlushRight      = teFlushRight
KControlBevelButtonAlignTextFlushLeft       = teFlushLeft
```

Bevel Button Text Placement

```
kControlBevelButtonPlaceSysDirection          = -1
kControlBevelButtonPlaceNormally              = 0
kControlBevelButtonPlaceToRightOfGraphic      = 1
kControlBevelButtonPlaceToLeftOfGraphic       = 2
kControlBevelButtonPlaceBelowGraphic          = 3
kControlBevelButtonPlaceAboveGraphic          = 4
```

Bevel Button Behavior

```
kControlBehaviorPushbutton                    = 0
kControlBehaviorToggles                       = 0x0100
kControlBehaviorSticky                        = 0x0200
kControlBehaviorSingleValueMenu               = 0
kControlBehaviorMultiValueMenu                = 0x4000
kControlBehaviorOffsetContents                = 0x8000
kControlBehaviorCommandMenu                   = 0x2000
```

Bevel Button Content Type

```
kControlContentTextOnly                       = 0
kControlContentIconSuiteRes                   = 1
kControlContentCIconRes                       = 2
kControlContentPictRes                        = 3
kControlContentIconSuiteHandle                = 129
kControlContentCIconHandle                    = 130
kControlContentPictHandle                     = 131
kControlContentIconRef                        = 132
```

Bevel Button Thickness

```
kControlBevelButtonSmallBevel                 = 0
kControlBevelButtonNormalBevel                = 1
kControlBevelButtonLargeBevel                 = 2
```

Bevel Button Menu Placement

```
kControlBevelButtonMenuOnBottom               = 0
kControlBevelButtonMenuOnRight                = (1 << 2)
```

Tab Size

```
kControlTabSizeLarge                          = 0
kControlTabSizeSmall                          = 1
```

Tab Direction

```
kControlTabDirectionNorth                     = 0
kControlTabDirectionSouth                     = 1
kControlTabDirectionEast                      = 2
kControlTabDirectionWest                      = 3
```

Clock Value Flag Constants

```
kControlClockNoFlags                          = 0
kControlClockIsDisplayOnly                    = 1
kControlClockIsLive                           = 2
```

Control Data Tags

```
kControlFontStyleTag                          = FOUR_CHAR_CODE('font')
```

```
kControlKeyFilterTag                            = FOUR_CHAR_CODE('fltr')
kControlBevelButtonContentTag                   = FOUR_CHAR_CODE('cont')
kControlBevelButtonTransformTag                 = FOUR_CHAR_CODE('tran')
kControlBevelButtonTextAlignTag                 = FOUR_CHAR_CODE('tali')
kControlBevelButtonTextOffsetTag                = FOUR_CHAR_CODE('toff')
kControlBevelButtonGraphicAlignTag              = FOUR_CHAR_CODE('gali')
kControlBevelButtonGraphicOffsetTag             = FOUR_CHAR_CODE('goff')
kControlBevelButtonTextPlaceTag                 = FOUR_CHAR_CODE('tplc')
kControlBevelButtonMenuValueTag                 = FOUR_CHAR_CODE('mval')
kControlBevelButtonMenuRefTag                   = FOUR_CHAR_CODE('mhnd')
kControlBevelButtonOwnedMenuRefTag              = FOUR_CHAR_CODE('omrf')
kControlBevelButtonKindTag                      = FOUR_CHAR_CODE('bebk')
kControlBevelButtonCenterPopupGlyphTag          = FOUR_CHAR_CODE('pglc')
kControlBevelButtonLastMenuTag                  = FOUR_CHAR_CODE('lmnu')
kControlBevelButtonMenuDelayTag                 = FOUR_CHAR_CODE('mdly')
kControlBevelButtonScaleIconTag                 = FOUR_CHAR_CODE('scal')
kControlTriangleLastValueTag                    = FOUR_CHAR_CODE('last')
kControlProgressBarIndeterminateTag             = FOUR_CHAR_CODE('inde')
kControlTabContentRectTag                       = FOUR_CHAR_CODE('rect')
kControlTabEnabledFlagTag                       = FOUR_CHAR_CODE('enab')
kControlTabFontStyleTag                         = kControlFontStyleTag
kControlTabInfoTag                              = FOUR_CHAR_CODE('tabi')
kControlGroupBoxMenuHandleTag                   = FOUR_CHAR_CODE('mhan')
kControlGroupBoxFontStyleTag                    = kControlFontStyleTag
kControlGroupBoxTitleRectTag                    = FOUR_CHAR_CODE('trec')
kControlImageWellContentTag                     = FOUR_CHAR_CODE('cont')
kControlImageWellTransformTag                   = FOUR_CHAR_CODE('tran')
kControlClockLongDateTag                        = FOUR_CHAR_CODE('date')
kControlClockFontStyleTag                       = kControlFontStyleTag
kControlClockAnimatingTag                       = FOUR_CHAR_CODE('anim')
kControlUserItemDrawProcTag                     = FOUR_CHAR_CODE('uidp')
kControlUserPaneDrawProcTag                     = FOUR_CHAR_CODE('draw')
kControlUserPaneHitTestProcTag                  = FOUR_CHAR_CODE('hitt')
kControlUserPaneTrackingProcTag                 = FOUR_CHAR_CODE('trak')
kControlUserPaneIdleProcTag                     = FOUR_CHAR_CODE('idle')
kControlUserPaneKeyDownProcTag                  = FOUR_CHAR_CODE('keyd')
kControlUserPaneActivateProcTag                 = FOUR_CHAR_CODE('acti')
kControlUserPaneFocusProcTag                    = FOUR_CHAR_CODE('foci')
kControlUserPaneBackgroundProcTag               = FOUR_CHAR_CODE('back')
kControlEditTextStyleTag                        = kControlFontStyleTag
kControlEditTextTextTag                         = FOUR_CHAR_CODE('text')
kControlEditTextTEHandleTag                     = FOUR_CHAR_CODE('than')
kControlEditTextKeyFilterTag                    = kControlKeyFilterTag,
kControlEditTextSelectionTag                    = FOUR_CHAR_CODE('sele')
kControlEditTextPasswordTag                     = FOUR_CHAR_CODE('pass')
kControlEditTextLockedTag                       = FOUR_CHAR_CODE('lock')
kControlEditTextValidationProcTag               = FOUR_CHAR_CODE('vali')
kControlStaticTextStyleTag                      = kControlFontStyleTag
kControlStaticTextTextTag                       = FOUR_CHAR_CODE('text')
kControlStaticTextTextHeightTag                 = FOUR_CHAR_CODE('thei')
kControlStaticTextTruncTag                      = FOUR_CHAR_CODE('trun')
kControlIconTransformTag                        = FOUR_CHAR_CODE('trfm')
kControlIconAlignmentTag                        = FOUR_CHAR_CODE('algn')
kControlIconResourceIDTag                       = FOUR_CHAR_CODE('ires')
kControlIconContentTag                          = FOUR_CHAR_CODE('cont')
kControlListBoxListHandleTag                    = FOUR_CHAR_CODE('lhan')
kControlListBoxKeyFilterTag                     = kControlKeyFilterTag
kControlListBoxFontStyleTag                     = kControlFontStyleTag
kControlListBoxDoubleClickTag                   = FOUR_CHAR_CODE('dblc')
kControlScrollTextBoxDelayBeforeAutoScrollTag   = FOUR_CHAR_CODE('stdl')
kControlScrollTextBoxDelayBetweenAutoScrollTag  = FOUR_CHAR_CODE('scdl')
kControlScrollTextBoxAutoScrollAmountTag        = FOUR_CHAR_CODE('samt')
kControlScrollTextBoxContentsTag                = FOUR_CHAR_CODE('tres')
kControlRoundButtonContentTag                   = FOUR_CHAR_CODE('cont'
kControlRoundButtonSizeTag                      = FOUR_CHAR_CODE('size'
```

Slider Orientation

```
kControlSliderPointsDownOrRight          = 0
kControlSliderPointsUpOrLeft             = 1
kControlSliderDoesNotPoint               = 2
```

Clock Control Type

```
kControlClockTypeHourMinute              = 0
kControlClockTypeHourMinuteSecond        = 1
kControlClockTypeMonthDayYear            = 2
kControlClockTypeMonthYear               = 3
```

Disclosure Triangle Orientation

```
kControlDisclosureTrianglePointDefault   = 0
kControlDisclosureTrianglePointRight     = 1
kControlDisclosureTrianglePointLeft      = 2
```

Pop-up Arrow Orientation

```
kControlPopupArrowOrientationEast        = 0
kControlPopupArrowOrientationWest        = 1
kControlPopupArrowOrientationNorth       = 2
kControlPopupArrowOrientationSouth       = 3
```

Key Filter Result Codes

```
kControlKeyFilterBlockKey                = 0
kControlKeyFilterPassKey                 = 1
```

Control Feature Bits

```
kControlSupportsGhosting                 = 1 << 0
kControlSupportsEmbedding                = 1 << 1
kControlSupportsFocus                    = 1 << 2
kControlWantsIdle                        = 1 << 3
kControlWantsActivate                    = 1 << 4
kControlHandlesTracking                  = 1 << 5
kControlSupportsDataAccess               = 1 << 6
kControlHasSpecialBackground             = 1 << 7
kControlGetsFocusOnClick                 = 1 << 8
kControlSupportsCalcBestRect             = 1 << 9
kControlSupportsLiveFeedback             = 1 << 10
kControlHasRadioBehavior                 = 1 << 11
```

Focusing Part Codes

```
KcontrolFocusNoPart                      = 0
KControlFocusNextPart                    = -1
kControlFocusPrevPart                    = -2
```

Disclosure Button State

```
kControlDisclosureButtonClosed           = 0
kControlDisclosureButtonDisclosed        = 1
```

Round Button Size

```
kControlRoundButtonNormalSize            = 0
kControlRoundButtonLargeSize             = 2
```

Control Feature Bit-Internal Timer

```
kControlIdlesWithTimer                          = 1 << 23
```

Control Kind (Mac OS X Only)

```
kControlKindBevelButton                         = FOUR_CHAR_CODE('bevl')
kControlKindImageWell                           = FOUR_CHAR_CODE('well')
kControlKindTabs                                = FOUR_CHAR_CODE('tabs')
kControlKindEditText                            = FOUR_CHAR_CODE('etxt')
kControlKindSlider                              = FOUR_CHAR_CODE('sldr')
kControlKindCheckGroupBox                       = FOUR_CHAR_CODE('cgrp')
kControlKindPopupGroupBox                       = FOUR_CHAR_CODE('pgrp')
kControlKindClock                               = FOUR_CHAR_CODE('clck')
kControlKindProgressBar                         = FOUR_CHAR_CODE('prgb')
kControlKindRelevanceBar                        = FOUR_CHAR_CODE('relb')
kControlKindLittleArrows                        = FOUR_CHAR_CODE('larr')
kControlKindDisclosureTriangle                  = FOUR_CHAR_CODE('dist')
kControlKindPicture                             = FOUR_CHAR_CODE('pict')
kControlKindIcon                                = FOUR_CHAR_CODE('icon')
kControlKindWindowHeader                        = FOUR_CHAR_CODE('whed')
kControlKindPlacard                             = FOUR_CHAR_CODE('plac')
kControlKindStaticText                          = FOUR_CHAR_CODE('stxt')
kControlKindSeparator                           = FOUR_CHAR_CODE('sepa')
kControlKindPopupArrow                          = FOUR_CHAR_CODE('parr')
kControlKindRadioGroup                          = FOUR_CHAR_CODE('rgrp')
kControlKindChasingArrows                       = FOUR_CHAR_CODE('carr')
kControlKindScrollingTextBox                    = FOUR_CHAR_CODE('stbx')
kControlKindDisclosureButton                    = FOUR_CHAR_CODE('disb')
kControlKindRoundButton                         = FOUR_CHAR_CODE('rndb')
```

Data Types

```
typedef SInt16 ControlFocusPart;
typedef SInt16 ControlKeyFilterResult;
typedef SInt16 ControlButtonGraphicAlignment;
typedef SInt16 ControlButtonTextAlignment;
typedef SInt16 ControlButtonTextPlacement;
typedef SInt16 ControlContentType;
typedef SInt16 ControlVariant;
typedef UInt16 ControlSliderOrientation;
typedef UInt16 ControlClockType;
typedef UInt16 ControlDisclosureTriangleOrientation;
typedef SInt16 ControlRoundButtonSize;
```

Button Content Info Structure

```
struct ControlButtonContentInfo
{
  ControlContentType  contentType;
  union {
    SInt16      resID;
    CIconHandle cIconHandle;
    Handle      iconSuite;
    IconRef     iconRef;
    PicHandle   picture;
    Handle      ICONHandle;
  }  u;
} ;
typedef struct ControlButtonContentInfo ControlButtonContentInfo;
typedef ControlButtonContentInfo *ControlButtonContentInfoPtr;
typedef ControlButtonContentInfo ControlImageContentInfo;
typedef ControlButtonContentInfo *ControlImageContentInfoPtr;
```

Tab Information Structures

```
struct ControlTabInfoRec
{
  SInt16 version;
  SInt16 iconSuiteID;
  Str255 name;
} ;
typedef struct ControlTabInfoRec ControlTabInfoRec;

struct ControlTabInfoRecV1
{
  SInt16      version;
  SInt16      iconSuiteID;
  CFStringRef name;
} ;
typedef struct ControlTabInfoRecV1 ControlTabInfoRecV1;
```

Tab Entry Structures

```
struct ControlTabEntry
{
  ControlButtonContentInfo *icon;
  CFStringRef               name;
  Boolean                   enabled;
} ;
typedef struct ControlTabEntry ControlTabEntry;
```

Edit Text Selection Structure

```
struct ControlEditTextSelectionRec
{
  SInt16 selStart;
  SInt16 selEnd;
} ;
typedef struct ControlEditTextSelectionRec ControlEditTextSelectionRec;
typedef ControlEditTextSelectionRec * ControlEditTextSelectionPtr;
```

Functions

Give Idle Time To Controls

```
void     IdleControls(WindowPtr inWindow);
```

Send Keyboard Event to Control with Keyboard Focus

```
Sint16    HandleControlKey(ControlRef inControl,SInt16 inKeyCode,SInt16 inCharCode,
          SInt16 inModifiers);
```

Set the Background for a Control

```
OSErr     SetUpControlBackground (ControlRef inControl,SInt16 inDepth,Boolean
inIsColorDevice);
```

KeyBoard Focus

```
OSErr     GetKeyboardFocus(WindowPtr inWindow,ControlRef *outControl);
OSErr     SetKeyboardFocus(WindowPtr inWindow,ControlRef inControl,ControlFocusPart inPart);
OSErr     AdvanceKeyboardFocus(WindowPtr inWindow);
OSErr     ReverseKeyboardFocus(WindowPtr inWindow);
OSErr     ClearKeyboardFocus(WindowPtr inWindow);
```

Control Features

```
OSErr     GetControlFeatures(ControlRef inControl,UInt32 *outFeatures);
```

Validating Controls

```
Boolean   IsValidControlhandle(ControlRef theControl);
```

Creating Controls Programmatically

```
OSStatus  CreateBevelButtonControl(WindowRef window,const Rect *boundsRect,
          CFStringRef title,ControlBevelThickness thickness,
          ControlBevelButtonBehavior behavior,ControlButtonContentInfoPtr info, SInt16
menuID,
          ControlBevelButtonMenuBehavior menuBehavior,
          ControlBevelButtonMenuPlacement menuPlacement,ControlRef *outControl);
OSStatus  CreateImageWellControl(WindowRef window,const Rect *boundsRect,
          const ControlButtonContentInfo *info,ControlRef *outControl);
OSStatus  CreateTabsControl(WindowRef window,const Rect *boundsRect,ControlTabSize size,
          ControlTabDirection direction,UInt16 numTabs,const ControlTabEntry *tabArray,
          ControlRef *outControl);
OSStatus  CreateEditTextControl(WindowRef window,const Rect *boundsRect, CFStringRef text,
          Boolean isPassword,Boolean useInlineInput,const ControlFontStyleRec *style,
          ControlRef *outControl);
OSStatus  CreateSliderControl(WindowRef window,const Rect *boundsRect,SInt32 value,
          SInt32 minimum,SInt32 maximum,ControlSliderOrientation  orientation,
          UInt16 numTickMarks,Boolean liveTracking,ControlActionUPP liveTrackingProc,
          ControlRef *outControl);
OSStatus  CreateGroupBoxControl(WindowRef window,const Rect *boundsRect, CFStringRef title,
          Boolean primary,ControlRef *outControl);
OSStatus  CreateCheckGroupBoxControl(WindowRef window,const Rect *boundsRect,
          CFStringRef title,Boolean primary,Boolean autoToggle,ControlRef *outControl);
OSStatus  CreatePopupGroupBoxControl(WindowRef window,const Rect *boundsRect,
          CFStringRef title,Boolean primary,SInt16 menuID,Boolean variableWidth,
          SInt16 titleWidth,SInt16 titleJustification,Style titleStyle,ControlRef
*outControl);
OSStatus  CreateClockControl(WindowRef window,const Rect *boundsRect,
          ControlClockType clockType,
          ControlClockFlags clockFlags,ControlRef *outControl);
OSStatus  CreateProgressBarControl(WindowRef window,const Rect *boundsRect,SInt32 value,
          SInt32 minimum,SInt32 maximum,Boolean indeterminate,ControlRef *outControl);
OSStatus  CreateLittleArrowsControl(WindowRef window,const Rect *boundsRect,SInt32 value,
          SInt32 minimum,SInt32 maximum,SInt32 increment,ControlRef *outControl);
OSStatus  CreateDisclosureTriangleControl(WindowRef window,const Rect *boundsRect,
          ControlDisclosureTriangleOrientation orientation, CFStringRef title,
          Boolean drawTitle,Boolean autoToggles,ControlRef *outControl);
OSStatus  CreatePictureControl(WindowRef window,const Rect *boundsRect,
          const ControlButtonContentInfo *content,Boolean dontTrack,ControlRef *outControl);
OSStatus  CreateIconControl(WindowRef window,const Rect *boundsRect,
          const ControlButtonContentInfo *icon,Boolean dontTrack,ControlRef *outControl);
OSStatus  CreateWindowHeaderControl(WindowRef window,const Rect *boundsRect,
          Boolean isListHeader,ControlRef *outControl);
OSStatus  CreatePlacardControl(WindowRef window,const Rect *boundsRect,ControlRef
*outControl);
OSStatus  CreateStaticTextControl(WindowRef window,const Rect *boundsRect,
          CFStringRef text,const ControlFontStyleRec *style,ControlRef *outControl);
OSStatus  CreateSeparatorControl(WindowRef window,const Rect *boundsRect,
          ControlRef *outControl);
OSStatus  CreatePopupArrowControl(WindowRef window,const Rect *boundsRect,
          ControlPopupArrowOrientation orientation,ControlPopupArrowSize size,
          ControlRef *outControl);
OSStatus  CreateRadioGroupControl(WindowRef window,const Rect *boundsRect,
          ControlRef *outControl);
OSStatus  CreateChasingArrowsControl(WindowRef window,const Rect *boundsRect,
          ControlRef *outControl);
```

```
OSStatus  CreateUserPaneControl(WindowRef window,const Rect *boundsRect,UInt32 features,
          ControlRef *outControl);
OSStatus  CreateScrollingTextBoxControl(WindowRef window,const Rect *boundsRect,
          SInt16 contentResID, Boolean autoScroll,UInt32 delayBeforeAutoScroll,
          UInt32 delayBetweenAutoScroll, UInt16 autoScrollAmount,ControlRef *outControl);
```

Creating Controls Programmatically (Controls Available on Mac OS X Only)

```
OSStatus  CreateRelevanceBarControl(WindowRef window,const Rect *boundsRect,SInt32 value,
          SInt32 minimum,SInt32 maximum,ControlRef *outControl);
OSStatus  CreateDisclosureButtonControl(WindowRef inWindow,const Rect *inBoundsRect,
          SInt32 inValue,Boolean inAutoToggles,ControlRef *  outControl);
OSStatus  CreateEditUnicodeTextControl(WindowRef window,const Rect *boundsRect,
          CFStringRef text,Boolean isPassword,const ControlFontStyleRec *style,
          ControlRef *outControl);
OSStatus  CreateRoundButtonControl(WindowRef nWindow,const Rect *inBoundsRect,
          ControlRoundButtonSize inSize,ControlButtonContentInfo *inContent,
          ControlRef *outControl);
```

Helper Functions

```
OSErr     GetBevelButtonMenuValue ControlRef inButton,SInt16 * outValue);
OSErr     SetBevelButtonMenuValue ControlRef inButton,SInt16 inValue);
OSErr     GetBevelButtonMenuHandle ControlRef inButton,MenuHandle * outHandle);
OSErr     GetBevelButtonContentInfo ControlRef inButton,
          ControlButtonContentInfoPtr outContent);
OSErr     SetBevelButtonContentInfo ControlRef inButton,ControlButtonContentInfoPtr
inContent);
OSErr     SetBevelButtonTransform ControlRef inButton,IconTransformType transform);
OSErr     SetBevelButtonGraphicAlignment ControlRef inButton,
          ControlButtonGraphicAlignment inAlign,SInt16 inHOffset,SInt16 inVOffset);
OSErr     SetBevelButtonTextAlignment ControlRef inButton,ControlButtonTextAlignment
inAlign,
          SInt16 inHOffset);
OSErr     SetBevelButtonTextPlacement ControlRef inButton,ControlButtonTextPlacement
inWhere);
OSErr     GetImageWellContentInfo(ControlRef inButton,ControlButtonContentInfoPtr
outContent);
OSErr     SetImageWellContentInfo(ControlRef inButton,ControlButtonContentInfoPtr
inContent);
OSErr     SetImageWellTransform(ControlRef inButton,IconTransformType inTransform);
OSErr     GetTabContentRect(ControlRef inTabControl,Rect * outContentRect);
OSErr     SetTabEnabled(ControlRef inTabControl,SInt16 inTabToHilite,Boolean inEnabled);
OSErr     SetDisclosureTriangleLastValue(ControlRef inTabControl,SInt16 inValue);
```

Application-Defined (Callback) Functions

```
ControlKeyFilterResult  myKeyFilterFunction(ControlRef controlRef,SInt16* keyCode,
                  SInt16 *charCode,EventModifiers *modifiers);
void                    myEditTextValidationFunction(ControlRef controlHdl);
```

Creating and Disposing of Universal Procedure Pointers-User Pane Functions

```
ControlUserPaneDrawUPP       NewControlUserPaneDrawUPP(ControlUserPaneDrawProcPtr
userRoutine);
ControlUserPaneHitTestUPP    NewControlUserPaneHitTestUPP(ControlUserPaneHitTestProcPtr
                             userRoutine);
ControlUserPaneTrackingUPP   NewControlUserPaneTrackingUPP(ControlUserPaneTrackingProcPtr
                             userRoutine);
ControlUserPaneIdleUPP       NewControlUserPaneIdleUPP(ControlUserPaneIdleProcPtr
                             userRoutine);
ControlUserPaneKeyDownUPP    NewControlUserPaneKeyDownUPP(ControlUserPaneKeyDownProcPtr
                             userRoutine);
ControlUserPaneActivateUPP   NewControlUserPaneActivateUPP(ControlUserPaneActivateProcPtr
```

```
                            userRoutine);
ControlUserPaneFocusUPP     NewControlUserPaneFocusUPP(ControlUserPaneFocusProcPtr
                            userRoutine);
ControlUserPaneBackgroundUPP NewControlUserPaneBackgroundUPP
(ControlUserPaneBackgroundProcPtr
                            userRoutine);
void      DisposeControlUserPaneDrawUPP(ControlUserPaneDrawUPP userUPP);
void      DisposeControlUserPaneHitTestUPP(ControlUserPaneHitTestUPP userUPP);
void      DisposeControlUserPaneTrackingUPP(ControlUserPaneTrackingUPP userUPP);
void      DisposeControlUserPaneIdleUPP(ControlUserPaneIdleUPP userUPP);
void      DisposeControlUserPaneKeyDownUPP(ControlUserPaneKeyDownUPP userUPP);
void      DisposeControlUserPaneActivateUPP(ControlUserPaneActivateUPP userUPP);
void      DisposeControlUserPaneFocusUPP(ControlUserPaneFocusUPP userUPP);
void      DisposeControlUserPaneBackgroundUPP(ControlUserPaneBackgroundUPP userUPP);
```

Demonstration Program Controls3 Listing

```
//
******************************************************************************************
// Controls3.h                                                          CLASSIC EVENT
MODEL
//
******************************************************************************************
//
// This program demonstrates the creation and handling of those controls not demonstrated in
// the programs Controls1 and Controls2 (Chapter 7), with the exception of List boxes
// and determinate progress bars.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  An 'MBAR' resource, and 'MENU' resources for Apple, File, Edit, and Demonstration
menus
//    (preload, non-purgeable).
//
// •  'MENU' resources (non-purgeable) for bevel button menus and for a pop-up group box.
//
// •  A 'WIND' resource (purgeable) (initially not visible).
//
// •  'DLOG' resources and associated 'DITL', 'dlgx' and 'dftb' resources (purgeable).
//
// •  'CNTL' resources (purgeable).
//
// •  A 'tab#' resource (purgeable.
//
// •  An icon family resource (purgeable).
//
// •  'PICT' resources (purgeable).
//
// •  'cicn' resources (purgeable).
//
// •  'STR#' resources (purgeable).
//
// •  'TEXT' and 'styl' resources (purgeable).
//
// •  'hrct' and an 'hwin' resources (preload, purgeable), which provide help balloons
//    describing  the various controls.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
//
******************************************************************************************

// ..........................................................................
includes

#include <Carbon.h>
#include <string.h>

// ..........................................................................
defines

#define rMenubar                128
#define mAppleApplication       128
#define  iAbout                 1
```

```
#define mFile                   129
#define  iQuit                  12
#define mDemonstration          131
#define  iBevelAndImage          1
#define  iTabEditClock          2
#define  iGroupArrowsProgress   3
#define  iSliders               4
#define  iTextBoxes             5
#define  iSmallControls         6
#define rBevelImageWindow       128
#define  cBevelButton1          128
#define  cBevelButton2          129
#define  cBevelButton3          130
#define  cBevelButton4          131
#define  cBevelButton5          132
#define  cBevelButton6          133
#define  cBevelButton7          134
#define  cBevelButton8          135
#define  cBevelButton9          136
#define  cBevelButton10         137
#define  cBevelButton11         138
#define  cBevelButton12         139
#define  cBevelButton13         140
#define  cBevelButton14         141
#define  cBevelButton15         142
#define  cBevelButton16         143
#define  cBevelButton17         144
#define  cBevelButton18         145
#define  cBevelButton19         146
#define  cBevelButton20         147
#define  cBevelButton21         148
#define  cImageWell1            149
#define  cImageWell2            150
#define  cPicture1              151
#define  cPicture2              152
#define  cColorIcon1           153
#define  cColorIcon2           154
#define  cIconSuite1            155
#define  cIconSuite2            156
#define  cWindowHeader          157
#define  rPartCodeStrings       128
#define  rGraphicAlignStrings   129
#define  rTextAlignStrings      130
#define  rTextPlacementStrings  131
#define rTabEditClockDialog     128
#define  iTabs                  2
#define  tabEditText            1
#define  tabClocks              2
#define  iEditTextUserPane      3
#define  iEditText1             5
#define  iEditText2             7
#define  iEditText3             9
#define  iExtractEditText       10
#define  iClocksUserPane        12
#define  iImageWellEditText     11
#define  iClocks1               14
#define  iClocks2               16
#define  iClocks3               18
#define  iExtractClocks         19
#define  iImageWellClocks       20
#define  kLeftArrow             0x1C
#define  kRightArrow            0x1D
#define  kUpArrow               0x1E
#define  kDownArrow             0x1F
#define  kBackspace             0x08
#define  kDelete                0x7F
```

```
#define rGroupArrowsProgDialog  129
#define  iCheckboxGroup         2
#define  iRadioGroupColor      3
#define  iStaticTextColorDepth 7
#define  iPopupGroup            8
#define  iUserPaneNamesInitials 9
#define  iRadioGroupNames       10
#define  iCheckboxShowInitials  13
#define  iUserPaneScoreAverage  15
#define  iRadioGroupScores      16
#define  iCheckboxShowAverages  19
#define  iStaticTextCache       26
#define  iLittleArrows          27
#define  iPushButtonExtract     28
#define  iImageWell             29
#define  iDisclosureTriangle    31
#define  iStaticTextDisclosure  32
#define  iProgressBar           34
#define rSlidersDialog          131
#define  iSlider1               2
#define  iSlider2               3
#define  iSlider3               4
#define  iSlider4               5
#define  iSlider1StaticText     9
#define  iSlider2StaticText     11
#define  iSlider3StaticText     13
#define  iSlider4StaticText     15
#define  iSlider5               17
#define  iUserPane1             18
#define  iSlider6               19
#define cSmallPopup             203
#define rSmallControlsString    136
#define rAboutDialog            132
#define MAX_UINT32              0xFFFFFFFF
#define MIN(a,b)                ((a) < (b) ? (a) : (b))

// ..............................................................................
typedefs

typedef struct
{
  ControlRef bevelButton1Ref;
  ControlRef bevelButton2Ref;
  ControlRef bevelButton3Ref;
  ControlRef bevelButton4Ref;
  ControlRef bevelButton5Ref;
  ControlRef bevelButton6Ref;
  ControlRef bevelButton7Ref;
  ControlRef bevelButton8Ref;
  ControlRef bevelButton9Ref;
  ControlRef bevelButton10Ref;
  ControlRef bevelButton11Ref;
  ControlRef bevelButton12Ref;
  ControlRef bevelButton13Ref;
  ControlRef bevelButton14Ref;
  ControlRef bevelButton15Ref;
  ControlRef bevelButton16Ref;
  ControlRef bevelButton17Ref;
  ControlRef bevelButton18Ref;
  ControlRef bevelButton19Ref;
  ControlRef bevelButton20Ref;
  ControlRef bevelButton21Ref;
  ControlRef imageWell1Ref;
  ControlRef imageWell2Ref;
  ControlRef picture1Ref;
  ControlRef picture2Ref;
```

```
  ControlRef colorIcon1Ref;
  ControlRef colorIcon2Ref;
  ControlRef iconSuite1Ref;
  ControlRef iconSuite2Ref;
  ControlRef windowHeaderRef;
}  BevelDocStruc;

typedef BevelDocStruc **BevelDocStrucHandle;
// ..................................................................... function
prototypes

void    main                          (void);
void    doPreliminaries               (void);
OSErr   quitAppEventHandler           (AppleEvent *,AppleEvent *,SInt32);
void    doGetControls                 (WindowRef);
void    doEvents                      (EventRecord *);
void    doMouseDown                   (EventRecord *);
void    doMenuChoice                  (SInt32);
void    doUpdate                      (EventRecord *);
void    doActivate                    (EventRecord *);
void    doActivateWindow              (WindowRef,Boolean);
void    doConcatPStrings              (Str255,Str255);
void    doCopyPString                 (Str255,Str255);

void    doBevelImagePictIcon          (void);
void    doBevelImagePictIconContent   (EventRecord *,WindowRef);
void    doDrawPartCode                (WindowRef,ControlRef,SInt16,SInt16);
void    doGraphicAlignment            (WindowRef,ControlRef,ControlRef);
void    doTextAlignment               (WindowRef,ControlRef,ControlRef);
void    doTextOffset                  (WindowRef,ControlRef,ControlRef);
void    doTextPlacement               (WindowRef,ControlRef,ControlRef);
void    doDrawMessage                 (WindowRef,Boolean);
void    doDrawLegends                 (WindowRef,Boolean);
void    helpTagsBevelImagePictIcon    (WindowRef);

void    doTabEditClock                (void);
void    doExtractEditText             (DialogRef);
void    doExtractDateTime             (DialogRef);
void    helpTagsTabEditClock          (DialogRef);

void    doGroupArrowsProgress         (void);
void    doCheckBoxGroupBox            (DialogRef);
void    doPopupGroupBox               (DialogRef);
void    doChasingAndProgress          (DialogRef);
void    doExtractCurrentStatus        (DialogRef);
void    helpTagsGroupArrowsProgress   (DialogRef);

void    doSliderUserPane              (void);
void    doDrawSliderValues            (DialogRef,ControlRef);
void    helpTagsSliders               (DialogRef);

void    doTextBox                     (void);

void    doSmallControls               (void);
void    doMouseDownSmallControls      (WindowRef,EventRecord *);

Boolean eventFilter                   (DialogRef,EventRecord *,SInt16 *);
void    editTextValidator             (ControlRef);
ControlKeyFilterResult  numericFilter (ControlRef,SInt16 *,SInt16 *,EventModifiers *);
void    arrowsActionFunction          (ControlRef,SInt16);
void    sliderActionFunction1         (ControlRef,SInt16);
void    sliderActionFunction2         (ControlRef,SInt16);
void    userPaneDrawFunction          (ControlRef,SInt16);
void    userPaneActivateFunction      (ControlRef,Boolean);
//
*******************************************************************************
```

```
// Controls3.c
//
********************************************************************************

// ..........................................................................
includes

#include "Controls3.h"

// ..................................................................... global
variables

Boolean gRunningOnX               = false;
Boolean gInBackground             = false;
Boolean gDone;
Str255  gCurrentString;
Boolean gBevelAndImageActive      = false;
Boolean gGroupArrowsProgressActive = false;
Boolean gSlidersActive            = false;

// ********************************************************************************
main

void  main(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  EventRecord   eventStructure;

  // ..................................................................... do
prelimiaries

  doPreliminaries();

  // ..................................................... set up menu bar and
menus

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    ExitToShell();
  SetMenuBar(menubarHdl);
  DrawMenuBar();

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
      DisableMenuItem(menuRef,0);
    }

    menuRef = GetMenuRef(mDemonstration);
    if(menuRef != NULL)
      EnableMenuItem(menuRef,iSmallControls);

    gRunningOnX = true;
  }
  // ..................................................................... enter
eventLoop

  gDone = false;
```

```
  while(!gDone)
  {
    if(WaitNextEvent(everyEvent,&eventStructure,MAX_UINT32,NULL))
      doEvents(&eventStructure);
  }
}

// **************************************************************************
doPreliminaries

void  doPreliminaries(void)
{
  OSErr osError;

  MoreMasterPointers(512);
  InitCursor();
  FlushEvents(everyEvent,0);

  osError = AEInstallEventHandler(kCoreEventClass,kAEQuitApplication,
                            NewAEEventHandlerUPP((AEEventHandlerProcPtr)
quitAppEventHandler),
                            0L,false);
  if(osError != noErr)
    ExitToShell();
}

// **************************************************************************
doQuitAppEvent

OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  OSErr    osError;
  DescType returnedType;
  Size     actualSize;

  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,
NULL,0,
                              &actualSize);
  if(osError == errAEDescNotFound)
  {
    gDone = true;
    osError = noErr;
  }
  else if(osError == noErr)
    osError = errAEParamMissed;

  return osError;
}

// **************************************************************************
doEvents
void  doEvents(EventRecord *eventStrucPtr)
{
  SInt32 menuChoice;
  SInt16 menuID, menuItem;

  switch(eventStrucPtr->what)
  {
    case kHighLevelEvent:
      AEProcessAppleEvent(eventStrucPtr);
      break;

    case keyDown:
      if((eventStrucPtr->modifiers & cmdKey) != 0)
      {
        menuChoice = MenuEvent(eventStrucPtr);
```

```
        menuID = HiWord(menuChoice);
        menuItem = LoWord(menuChoice);
        if(menuID == mFile && menuItem  == iQuit)
          gDone = true;
      }
      break;

    case mouseDown:
      doMouseDown(eventStrucPtr);
      break;

    case updateEvt:
      doUpdate(eventStrucPtr);
      break;

    case activateEvt:
      doActivate(eventStrucPtr);
      break;

    case osEvt:
      switch((eventStrucPtr->message >> 24) & 0x000000FF)
      {
        case suspendResumeMessage:
          if((eventStrucPtr->message & resumeFlag) == 1)
          {
            SetThemeCursor(kThemeArrowCursor);
            gInBackground = false;
          }
          else
            gInBackground = true;
      }
      break;
  }
}
// ****************************************************************************
doMouseDown

void  doMouseDown(EventRecord *eventStrucPtr)
{
  WindowPartCode partCode;
  WindowRef      windowRef;
  MenuRef        menuRef;
  WindowClass    windowClass;

  partCode = FindWindow(eventStrucPtr->where,&windowRef);

  switch(partCode)
  {
    case inMenuBar:
      menuRef = GetMenuRef(mDemonstration);
      if(gBevelAndImageActive)
        DisableMenuItem(menuRef,iBevelAndImage);
      else
        EnableMenuItem(menuRef,iBevelAndImage);
      doMenuChoice(MenuSelect(eventStrucPtr->where));
      break;

    case inContent:
      GetWindowClass(windowRef,&windowClass);
      if(windowClass == kFloatingWindowClass)
        doMouseDownSmallControls(windowRef,eventStrucPtr);
      else if(windowRef != FrontNonFloatingWindow())
        SelectWindow(windowRef);
      else
      {
        if(gBevelAndImageActive)
```

```
          doBevelImagePictIconContent(eventStrucPtr,windowRef);
      }
      break;

    case inDrag:
      DragWindow(windowRef,eventStrucPtr->where,NULL);
      break;

    case inGoAway:
      if(TrackGoAway(windowRef,eventStrucPtr->where) == true)
      {
        DisposeWindow(windowRef);
        gBevelAndImageActive = false;
      }
      break;

  }
}
// *****************************************************************************
doMenuChoice

void  doMenuChoice(SInt32 menuChoice)
{
  MenuID        menuID;
  MenuItemIndex menuItem;
  MenuRef       menuRef;

  menuID = HiWord(menuChoice);
  menuItem = LoWord(menuChoice);

  if(menuID == 0)
    return;

  switch(menuID)
  {
    case mAppleApplication:
      if(menuItem == iAbout)
        SysBeep(10);
      break;

    case mFile:
      if(menuItem == iQuit)
        gDone = true;
      break;

    case mDemonstration:
      switch(menuItem)
      {
        case iBevelAndImage:
          gBevelAndImageActive = true;
          doBevelImagePictIcon();
          break;

        case iTabEditClock:
          doTabEditClock();
          break;

        case iGroupArrowsProgress:
          gGroupArrowsProgressActive = true;
          doGroupArrowsProgress();
          break;

        case iSliders:
          gSlidersActive = true;
          doSliderUserPane();
          break;
```

```
        case iTextBoxes:
          doTextBox();
          break;
        case iSmallControls:
          doSmallControls();
          menuRef = GetMenuRef(mDemonstration);
          DisableMenuItem(menuRef,iSmallControls);
          break;
      }
      break;
  }

  HiliteMenu(0);
}

// **************************************************************************************
doUpdate

void  doUpdate(EventRecord *eventStrucPtr)
{
  WindowRef windowRef;
  RgnHandle visibleRegionHdl = NewRgn();
  Boolean   drawMode = false;

  windowRef = (WindowRef) eventStrucPtr->message;

  BeginUpdate(windowRef);

  SetPortWindowPort(windowRef);

  GetPortVisibleRegion(GetWindowPort(windowRef),visibleRegionHdl);
  UpdateControls(windowRef,visibleRegionHdl);

  if(gBevelAndImageActive)
  {
    if(windowRef == FrontNonFloatingWindow())
    {
      doDrawMessage(windowRef,!gInBackground);
      doDrawLegends(windowRef,!gInBackground);
    }
    else
    {
      doDrawMessage(windowRef,gInBackground);
      doDrawLegends(windowRef,gInBackground);
    }
  }

  EndUpdate(windowRef);
}

// **************************************************************************************
doActivate

void  doActivate(EventRecord *eventStrucPtr)
{
  WindowRef windowRef;
  Boolean   becomingActive;
  windowRef = (WindowRef) eventStrucPtr->message;
  becomingActive = ((eventStrucPtr->modifiers & activeFlag) == activeFlag);

  doActivateWindow(windowRef,becomingActive);
}

// **************************************************************************************
doActivateWindow
```

```
void  doActivateWindow(WindowRef windowRef,Boolean becomingActive)
{
  ControlRef controlRef;

  GetRootControl(windowRef,&controlRef);

  if(becomingActive)
  {
    if(gBevelAndImageActive)
    {
      ActivateControl(controlRef);
      doDrawMessage(windowRef,becomingActive);
      doDrawLegends(windowRef,becomingActive);
    }
  }
  else
  {
    if(gBevelAndImageActive)
    {
      DeactivateControl(controlRef);
      doDrawMessage(windowRef,becomingActive);
      doDrawLegends(windowRef,becomingActive);
    }
  }
}

// ************************************************************************
doConcatPStrings

void  doConcatPStrings(Str255 targetString,Str255 appendString)
{
  SInt16 appendLength;

  appendLength = MIN(appendString[0],255 - targetString[0]);

  if(appendLength > 0)
  {
    BlockMoveData(appendString+1,targetString+targetString[0]+1,(SInt32) appendLength);
    targetString[0] += appendLength;
  }
}

// ************************************************************************
doCopyPString

void  doCopyPString(Str255 sourceString,Str255 destinationString)
{
  SInt16 stringLength;
  stringLength = sourceString[0];
  BlockMove(sourceString + 1,destinationString + 1,stringLength);
  destinationString[0] = stringLength;
}

//
****************************************************************************************
// BevelImagePictIcon.c

//
****************************************************************************************

// ..........................................................................
includes

#include "Controls3.h"
```

```
// ..................................................................... global
variables

WindowRef gWindowRef;

extern Boolean gRunningOnX;
extern Str255  gCurrentString;
extern Boolean gInBackground;

// ********************************************************************
doBevelImagePictIcon

void  doBevelImagePictIcon(void)
{
  BevelDocStrucHandle bevelDocStrucHdl;

  // ............................................. initial advisory text for window
header

  doCopyPString("\pBalloon (OS 8/9) and Help tag (OS X) help is available",gCurrentString);

  // .............. open a window, set font size, set theme-compliant color/pattern for
window

  if(!(gWindowRef = GetNewCWindow(rBevelImageWindow,NULL,(WindowRef)-1)))
    ExitToShell();

  SetPortWindowPort(gWindowRef);
  UseThemeFont(kThemeSmallSystemFont,smSystemScript);

  SetThemeWindowBackground(gWindowRef,kThemeBrushDialogBackgroundActive,true);

  // ........... get block for document structure, assign handle to window record refCon
field

  if(!(bevelDocStrucHdl = (BevelDocStrucHandle) NewHandle(sizeof(BevelDocStruc))))
    ExitToShell();
  SetWRefCon(gWindowRef,(SInt32) bevelDocStrucHdl);

  // ..................................... get controls, help tags if OS X, and show
window

  doGetControls(gWindowRef);

  if(gRunningOnX)
    helpTagsBevelImagePictIcon(gWindowRef);
  ShowWindow(gWindowRef);
}

// ********************************************************************************
doGetControls

void  doGetControls(WindowRef windowRef)
{
  ControlRef                controlRef;
  BevelDocStrucHandle       bevelDocStrucHdl;
  ThemeButtonKind           buttonKind = kThemeRoundedBevelButton;
  ControlButtonTextPlacement textPlacementBelow = kControlBevelButtonPlaceBelowGraphic;
  ControlButtonTextPlacement textPlacementAbove = kControlBevelButtonPlaceAboveGraphic;
  Boolean                   centerPopupGlyph = true;
  ControlFontStyleRec       controlFontStyleStruc;

  // ............... create root control for window, get handle to window's document
structure

  if(!gRunningOnX)
```

```
    CreateRootControl(windowRef,&controlRef);
  bevelDocStrucHdl = (BevelDocStrucHandle) (GetWRefCon(windowRef));

  // ..................................................................... get the
controls

  if(!((*bevelDocStrucHdl)->bevelButton1Ref = GetNewControl(cBevelButton1,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->bevelButton2Ref = GetNewControl(cBevelButton2,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->bevelButton3Ref = GetNewControl(cBevelButton3,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->bevelButton4Ref = GetNewControl(cBevelButton4,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->bevelButton5Ref = GetNewControl(cBevelButton5,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->bevelButton6Ref = GetNewControl(cBevelButton6,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->bevelButton7Ref = GetNewControl(cBevelButton7,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->bevelButton8Ref = GetNewControl(cBevelButton8,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->bevelButton9Ref = GetNewControl(cBevelButton9,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->bevelButton10Ref = GetNewControl(cBevelButton10,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->bevelButton11Ref = GetNewControl(cBevelButton11,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->bevelButton12Ref = GetNewControl(cBevelButton12,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->bevelButton13Ref = GetNewControl(cBevelButton13,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->bevelButton14Ref = GetNewControl(cBevelButton14,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->bevelButton15Ref = GetNewControl(cBevelButton15,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->bevelButton16Ref = GetNewControl(cBevelButton16,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->bevelButton17Ref = GetNewControl(cBevelButton17,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->bevelButton18Ref = GetNewControl(cBevelButton18,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->bevelButton19Ref = GetNewControl(cBevelButton19,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->bevelButton20Ref = GetNewControl(cBevelButton20,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->bevelButton21Ref = GetNewControl(cBevelButton21,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->imageWell1Ref = GetNewControl(cImageWell1,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->imageWell2Ref = GetNewControl(cImageWell2,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->picture1Ref = GetNewControl(cPicture1,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->picture2Ref = GetNewControl(cPicture2,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->colorIcon1Ref = GetNewControl(cColorIcon1,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->colorIcon2Ref = GetNewControl(cColorIcon2,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->iconSuite1Ref= GetNewControl(cIconSuite1,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->iconSuite2Ref= GetNewControl(cIconSuite2,windowRef)))
    ExitToShell();
  if(!((*bevelDocStrucHdl)->windowHeaderRef = GetNewControl(cWindowHeader,windowRef)))
    ExitToShell();
```

```
  // ............................. if running on OS X, make first four bevel buttons
rounded

  if(gRunningOnX)
  {
    SetControlData((*bevelDocStrucHdl)->bevelButton1Ref,kControlEntireControl,
                   kControlBevelButtonKindTag,sizeof(buttonKind),&buttonKind);
    SetControlData((*bevelDocStrucHdl)->bevelButton2Ref,kControlEntireControl,
                   kControlBevelButtonKindTag,sizeof(buttonKind),&buttonKind);
    SetControlData((*bevelDocStrucHdl)->bevelButton3Ref,kControlEntireControl,
                   kControlBevelButtonKindTag,sizeof(buttonKind),&buttonKind);
    SetControlData((*bevelDocStrucHdl)->bevelButton4Ref,kControlEntireControl,
                   kControlBevelButtonKindTag,sizeof(buttonKind),&buttonKind);
  }

  // ..................................... set text placement for 2nd and 21st bevel
button

  SetControlData((*bevelDocStrucHdl)->bevelButton2Ref,kControlEntireControl,
                 kControlBevelButtonTextPlaceTag,sizeof(textPlacementBelow),
                 &textPlacementBelow);
  SetControlData((*bevelDocStrucHdl)->bevelButton21Ref,kControlEntireControl,
                 kControlBevelButtonTextPlaceTag,sizeof(textPlacementAbove),
                 &textPlacementAbove);
  // ..................................... set position of pop-up arrow in 6th bevel
button

  SetControlData((*bevelDocStrucHdl)->bevelButton6Ref,kControlEntireControl,
                 kControlBevelButtonCenterPopupGlyphTag,sizeof(centerPopupGlyph),
                 &centerPopupGlyph);

  // ............................. set font for 20th bevel button to small bold system
font

  controlFontStyleStruc.flags = kControlUseFontMask;
  controlFontStyleStruc.font = kControlFontSmallBoldSystemFont;
  SetControlFontStyle((*bevelDocStrucHdl)->bevelButton20Ref,&controlFontStyleStruc);

  // ........................................... set 3rd bevel button to the mixed
state

  SetControlValue((*bevelDocStrucHdl)->bevelButton3Ref,2);
}

// ************************************************************
doBevelImagePictIconContent

void  doBevelImagePictIconContent(EventRecord *eventStrucPtr,WindowRef windowRef)
{
  BevelDocStrucHandle bevelDocStrucHdl;
  ControlRef          controlRef;
  SInt16              partCode, menuItem;

  bevelDocStrucHdl = (BevelDocStrucHandle) (GetWRefCon(windowRef));

  SetPortWindowPort(windowRef);
  GlobalToLocal(&eventStrucPtr->where);
  partCode = FindControl(eventStrucPtr->where,windowRef,&controlRef);

  doDrawPartCode(windowRef,(*bevelDocStrucHdl)->windowHeaderRef,partCode,1234);

  if(partCode)
  {
    partCode = TrackControl(controlRef,eventStrucPtr->where,NULL);
```

```
    if(controlRef == (*bevelDocStrucHdl)->bevelButton1Ref ||
       controlRef == (*bevelDocStrucHdl)->bevelButton2Ref ||
       controlRef == (*bevelDocStrucHdl)->bevelButton3Ref ||
       controlRef == (*bevelDocStrucHdl)->bevelButton4Ref ||
       controlRef == (*bevelDocStrucHdl)->bevelButton9Ref ||
       controlRef == (*bevelDocStrucHdl)->bevelButton13Ref ||
       controlRef == (*bevelDocStrucHdl)->bevelButton14Ref ||
       controlRef == (*bevelDocStrucHdl)->bevelButton15Ref ||
       controlRef == (*bevelDocStrucHdl)->bevelButton16Ref ||
       controlRef == (*bevelDocStrucHdl)->bevelButton17Ref)
    {
      doDrawPartCode(windowRef,(*bevelDocStrucHdl)->windowHeaderRef,partCode,4321);
    }
    else if(controlRef == (*bevelDocStrucHdl)->bevelButton5Ref ||
            controlRef == (*bevelDocStrucHdl)->bevelButton6Ref ||
            controlRef == (*bevelDocStrucHdl)->bevelButton7Ref ||
            controlRef == (*bevelDocStrucHdl)->bevelButton8Ref)
    {
      if(partCode == kControlMenuPart)
      {
        GetBevelButtonMenuValue(controlRef,&menuItem);
        doDrawPartCode(windowRef,(*bevelDocStrucHdl)->windowHeaderRef,partCode,menuItem);
      }
      else
        doDrawPartCode(windowRef,(*bevelDocStrucHdl)->windowHeaderRef,partCode,4321);
    }
    else if(controlRef == (*bevelDocStrucHdl)->bevelButton10Ref ||
            controlRef == (*bevelDocStrucHdl)->bevelButton11Ref ||
            controlRef == (*bevelDocStrucHdl)->bevelButton12Ref)
    {
      if(partCode != kControlEntireControl)
      {
        SetControlValue((*bevelDocStrucHdl)->bevelButton10Ref,0);
        SetControlValue((*bevelDocStrucHdl)->bevelButton11Ref,0);
        SetControlValue((*bevelDocStrucHdl)->bevelButton12Ref,0);
        SetControlValue(controlRef,1);
      }
      doDrawPartCode(windowRef,(*bevelDocStrucHdl)->windowHeaderRef,partCode,4321);
    }
    else if(controlRef == (*bevelDocStrucHdl)->bevelButton18Ref)
    {
      doDrawPartCode(windowRef,(*bevelDocStrucHdl)->windowHeaderRef,partCode,4321);
      doGraphicAlignment(windowRef,controlRef,(*bevelDocStrucHdl)->windowHeaderRef);
    }
    else if(controlRef == (*bevelDocStrucHdl)->bevelButton19Ref)
    {
      doDrawPartCode(windowRef,(*bevelDocStrucHdl)->windowHeaderRef,partCode,4321);
      doTextAlignment(windowRef,controlRef,(*bevelDocStrucHdl)->windowHeaderRef);
    }
    else if(controlRef == (*bevelDocStrucHdl)->bevelButton20Ref)
    {
      doDrawPartCode(windowRef,(*bevelDocStrucHdl)->windowHeaderRef,partCode,4321);
      doTextOffset(windowRef,controlRef,(*bevelDocStrucHdl)->windowHeaderRef);
    }
    else if(controlRef == (*bevelDocStrucHdl)->bevelButton21Ref)
    {
      doDrawPartCode(windowRef,(*bevelDocStrucHdl)->windowHeaderRef,partCode,4321);
      doTextPlacement(windowRef,controlRef,(*bevelDocStrucHdl)->windowHeaderRef);
    }
    else
      doDrawPartCode(windowRef,(*bevelDocStrucHdl)->windowHeaderRef,partCode,4321);
  }
}

// ********************************************************************************
doDrawPartCode
```

```
void  doDrawPartCode(WindowRef windowRef,ControlRef windowHeaderRef,SInt16 partCode,
                     SInt16 menuItem)

{
  SInt16 stringIndex;
  Str255 theString;
  if(partCode == kControlEntireControl)
    stringIndex = 1;
  else if(partCode == kControlMenuPart)
    stringIndex = 2;
  else if(partCode == kControlTrianglePart)
    stringIndex = 3;
  else if(partCode == kControlEditTextPart)
    stringIndex = 4;
  else if(partCode == kControlPicturePart)
    stringIndex = 5;
  else if(partCode == kControlIconPart)
    stringIndex = 6;
  else if(partCode == kControlClockPart)
    stringIndex = 7;
  else if(partCode == kControlListBoxPart)
    stringIndex = 8;
  else if(partCode == kControlListBoxDoubleClickPart)
    stringIndex = 9;
  else if(partCode == kControlImageWellPart)
    stringIndex = 10;
  else if(partCode == kControlRadioGroupPart)
    stringIndex = 11;
  else if(partCode == kControlButtonPart)
    stringIndex = 12;
  else if(partCode == kControlIndicatorPart)
    stringIndex = 13;

  if(menuItem > 0 && menuItem < 1234)
  {
    doCopyPString("\pTrackControl returned ",gCurrentString);
    GetIndString(theString,rPartCodeStrings,stringIndex);
    doConcatPStrings(gCurrentString,theString);
    doConcatPStrings(gCurrentString,"\p    GetBevelButtonMenuValue returned menu item ");
    NumToString((SInt32) menuItem,theString);
    doConcatPStrings(gCurrentString,theString);
  }
  else if(menuItem == 1234 || menuItem == 4321)
  {
    if(menuItem == 1234)
      doCopyPString("\pMouse-down in ",gCurrentString);
    else if(menuItem == 4321)
      doCopyPString("\pTrackControl returned ",gCurrentString);
    GetIndString(theString,rPartCodeStrings,stringIndex);
    doConcatPStrings(gCurrentString,theString);
  }

  Draw1Control(windowHeaderRef);
  doDrawMessage(windowRef,true);
}

// ************************************************************************
doGraphicAlignment
void  doGraphicAlignment(WindowRef windowRef,ControlRef controlRef,ControlRef
windowHeaderRef)
{
  SInt16                        a, b;
  UInt32                        finalTicks;
  ControlButtonGraphicAlignment alignmentConstant = 0;
```

```
  if(!gRunningOnX)
    SetThemeCursor(kThemeWatchCursor);

  for(a=1;a<10;a++)
  {
    Delay(60,&finalTicks);
    alignmentConstant++;
    if(alignmentConstant == 9)
      alignmentConstant = 0;

    Draw1Control(windowHeaderRef);
    GetIndString(gCurrentString,rGraphicAlignStrings,a);
    doDrawMessage(windowRef,true);

    for(b=0;b<=52;b++)
    {
      SetBevelButtonGraphicAlignment(controlRef,alignmentConstant,b,b);
      Delay(2,&finalTicks);
      Draw1Control(controlRef);
      QDFlushPortBuffer(GetWindowPort(FrontNonFloatingWindow()),NULL);
    }
  }

  if(!gRunningOnX)
    SetThemeCursor(kThemeArrowCursor);
}

// ************************************************************************
doTextAlignment

void  doTextAlignment(WindowRef windowRef,ControlRef controlRef,ControlRef windowHeaderRef)
{
  SInt16                    a, b;
  UInt32                    finalTicks;
  ControlButtonTextAlignment alignmentConstant = -3;

  if(!gRunningOnX)
    SetThemeCursor(kThemeWatchCursor);

  for(a=1;a<4;a++)
  {
    Delay(60,&finalTicks);
    alignmentConstant++;
    if(alignmentConstant == 0)
      alignmentConstant++;

    Draw1Control(windowHeaderRef);
    GetIndString(gCurrentString,rTextAlignStrings,a);
    doDrawMessage(windowRef,true);
    for(b=0;b<=40;b++)
    {
      SetBevelButtonTextAlignment(controlRef,alignmentConstant,b);
      Delay(2,&finalTicks);
      Draw1Control(controlRef);
      QDFlushPortBuffer(GetWindowPort(FrontNonFloatingWindow()),NULL);
    }
  }

  if(!gRunningOnX)
    SetThemeCursor(kThemeArrowCursor);
}

// ************************************************************************
doTextOffset

void  doTextOffset(WindowRef windowRef,ControlRef controlRef,ControlRef windowHeaderRef)
```

```
{
  ControlButtonTextAlignment alignmentConstant;
  SInt16                     offset;
  UInt32                     finalTicks;

  if(!gRunningOnX)
    SetThemeCursor(kThemeWatchCursor);

  Draw1Control(windowHeaderRef);
  doCopyPString("\pOffset from left",gCurrentString);
  doDrawMessage(windowRef,true);

  alignmentConstant = kControlBevelButtonAlignTextFlushLeft;
  SetControlData(controlRef,kControlEntireControl,kControlBevelButtonTextAlignTag,
                 sizeof(alignmentConstant),&alignmentConstant);
  Draw1Control(controlRef);

  for(offset=1;offset<27;offset++)
  {
    Delay(15,&finalTicks);
    SetControlData(controlRef,kControlEntireControl,kControlBevelButtonTextOffsetTag,
                   sizeof(offset),&offset);
    Draw1Control(controlRef);
    QDFlushPortBuffer(GetWindowPort(FrontNonFloatingWindow()),NULL);
  }

  Delay(60,&finalTicks);

  Draw1Control(windowHeaderRef);
  doCopyPString("\pOffset from right",gCurrentString);
  doDrawMessage(windowRef,true);

  alignmentConstant = kControlBevelButtonAlignTextFlushRight;
  SetControlData(controlRef,kControlEntireControl,kControlBevelButtonTextAlignTag,
                 sizeof(alignmentConstant),&alignmentConstant);
  for(offset=0;offset<13;offset++)
  {
    Delay(15,&finalTicks);
    SetControlData(controlRef,kControlEntireControl,kControlBevelButtonTextOffsetTag,
                   sizeof(offset),&offset);
    Draw1Control(controlRef);
    QDFlushPortBuffer(GetWindowPort(FrontNonFloatingWindow()),NULL);
  }

  if(!gRunningOnX)
    SetThemeCursor(kThemeArrowCursor);
}

// ************************************************************************
doTextPlacement

void  doTextPlacement(WindowRef windowRef,ControlRef controlRef,ControlRef windowHeaderRef)
{
  ControlButtonTextPlacement placementConstant;
  UInt32                     finalTicks;

  if(!gRunningOnX)
    SetThemeCursor(kThemeWatchCursor);

  for(placementConstant = 1;placementConstant < 5;placementConstant++)
  {
    Delay(60,&finalTicks);
    SetBevelButtonTextPlacement(controlRef,placementConstant);
    Draw1Control(controlRef);
    Draw1Control(windowHeaderRef);
    GetIndString(gCurrentString,rTextPlacementStrings,placementConstant);
```

```
    doDrawMessage(windowRef,true);
    QDFlushPortBuffer(GetWindowPort(FrontNonFloatingWindow()),NULL);
  }

  if(!gRunningOnX)
    SetThemeCursor(kThemeArrowCursor);
}

// ************************************************************************
doDrawMessage

void  doDrawMessage(WindowRef windowRef,Boolean inState)
{
  Rect        portRect;
  CFStringRef stringRef;
  Rect        textBoxRect;

  if(windowRef == gWindowRef)
  {
    SetPortWindowPort(windowRef);

    GetWindowPortBounds(windowRef,&portRect);
    stringRef = CFStringCreateWithPascalString(NULL,gCurrentString,
kCFStringEncodingMacRoman);
    SetRect(&textBoxRect,portRect.left,7,portRect.right,22);

    if(inState == kThemeStateActive)
      TextMode(srcOr);
    else
      TextMode(grayishTextOr);

    DrawThemeTextBox(stringRef,kThemeSmallSystemFont,inState,true,&textBoxRect,teJustCenter,
                     NULL);
    if(stringRef != NULL)
      CFRelease(stringRef);
  }
}

// ************************************************************************
doDrawLegends

void  doDrawLegends(WindowRef windowRef,Boolean inState)
{
  Rect theRect;

  if(windowRef == gWindowRef)
  {
    SetPortWindowPort(windowRef);

    if(gRunningOnX)
    {
      SetRect(&theRect,30,54,585,67);
      EraseRect(&theRect);
      OffsetRect(&theRect,0,91);
      EraseRect(&theRect);
      OffsetRect(&theRect,0,92);
      EraseRect(&theRect);
    }

    if(inState == kThemeStateActive)
      TextMode(srcOr);
    else
      TextMode(grayishTextOr);

    SetRect(&theRect,30,53,220,68);
    if(gRunningOnX)
```

```
      DrawThemeTextBox(CFSTR("Bevel button content"),kThemeSmallSystemFont,inState,false,
                       &theRect,teJustLeft,NULL);
    else
      DrawThemeTextBox(CFSTR("Bevel sizes and bevel button content"),kThemeSmallSystemFont,
                       inState,false,&theRect,teJustLeft,NULL);
    SetRect(&theRect,313,53,513,68);
    DrawThemeTextBox(CFSTR("Menu position and behavior"),kThemeSmallSystemFont,inState,
                     false,&theRect,teJustLeft,NULL);
    SetRect(&theRect,30,144,180,159);
    DrawThemeTextBox(CFSTR("Bevel button behavior"),kThemeSmallSystemFont,inState,false,
                     &theRect,teJustLeft,NULL);
    SetRect(&theRect,313,144,513,159);
    DrawThemeTextBox(CFSTR("Graphic/text alignment & offset"),kThemeSmallSystemFont,inState,
                     false,&theRect,teJustLeft,NULL);
    SetRect(&theRect,490,144,640,159);
    DrawThemeTextBox(CFSTR("Text placement"),kThemeSmallSystemFont,inState,false,&theRect,
                     teJustLeft,NULL);
    SetRect(&theRect,30,235,180,250);
    DrawThemeTextBox(CFSTR("Image wells"),kThemeSmallSystemFont,inState,false,&theRect,
                     teJustLeft,NULL);
    SetRect(&theRect,168,235,308,250);
    DrawThemeTextBox(CFSTR("Picture controls"),kThemeSmallSystemFont,inState,false,&theRect,
                     teJustLeft,NULL);
    SetRect(&theRect,313,235,463,250);
    DrawThemeTextBox(CFSTR("Icon controls (cicn)"),kThemeSmallSystemFont,inState,false,
                     &theRect,teJustLeft,NULL);
    SetRect(&theRect,451,235,601,250);
    DrawThemeTextBox(CFSTR("Icon controls (icon suite)"),kThemeSmallSystemFont,inState,
false,
                     &theRect,teJustLeft,NULL);
  }
}

// ************************************************************
helpTagsBevelImagePictIcon

void  helpTagsBevelImagePictIcon(WindowRef windowRef)
{
  BevelDocStrucHandle bevelDocStrucHdl;
  HMHelpContentRec    helpContent;

  bevelDocStrucHdl = (BevelDocStrucHandle) (GetWRefCon(windowRef));

  memset(&helpContent,0,sizeof(helpContent));
  HMSetTagDelay(500);
  HMSetHelpTagsDisplayed(true);

  helpContent.version = kMacHelpVersion;
  helpContent.tagSide = kHMOutsideTopCenterAligned;
  helpContent.content[kHMMinimumContentIndex].contentType = kHMStringResContent;
  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmResID = 132;

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 1;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton1Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 2;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton2Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 3;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton3Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 4;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton4Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 5;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton5Ref,&helpContent);
```

```
helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 6;
HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton6Ref,&helpContent);

helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 7;
HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton7Ref,&helpContent);

helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 8;
HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton8Ref,&helpContent);

helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 9;
HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton9Ref,&helpContent);

helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 10;
HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton10Ref,&helpContent);

helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 10;
HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton11Ref,&helpContent);

helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 10;
HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton12Ref,&helpContent);

helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 11;
HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton13Ref,&helpContent);

helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 11;
HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton14Ref,&helpContent);

helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 11;
HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton15Ref,&helpContent);

helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 12;
HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton16Ref,&helpContent);

helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 13;
HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton17Ref,&helpContent);

helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 14;
HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton18Ref,&helpContent);

helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 15;
HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton19Ref,&helpContent);

helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 16;
HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton20Ref,&helpContent);

helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 17;
HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton21Ref,&helpContent);

helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 18;
HMSetControlHelpContent((*bevelDocStrucHdl)->imageWell1Ref,&helpContent);

helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 19;
HMSetControlHelpContent((*bevelDocStrucHdl)->imageWell2Ref,&helpContent);
helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 20;
HMSetControlHelpContent((*bevelDocStrucHdl)->picture1Ref,&helpContent);

helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 21;
HMSetControlHelpContent((*bevelDocStrucHdl)->picture2Ref,&helpContent);

helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 22;
HMSetControlHelpContent((*bevelDocStrucHdl)->colorIcon1Ref,&helpContent);

helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 23;
HMSetControlHelpContent((*bevelDocStrucHdl)->colorIcon2Ref,&helpContent);

helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 24;
```

```
  HMSetControlHelpContent((*bevelDocStrucHdl)->iconSuite1Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 25;
  HMSetControlHelpContent((*bevelDocStrucHdl)->iconSuite2Ref,&helpContent);

}

//
**********************************************************************************************
// TabEditClock.c
//
**********************************************************************************************

// ..........................................................................
includes

#include "Controls3.h"

// ..................................................................... global
variables

extern Boolean  gRunningOnX;

// ************************************************************************
doTabEditClock

void  doTabEditClock(void)
{
  DialogRef                   dialogRef;
  ControlRef                  controlRef;
  Str255                      initialName = "\pYour name here";
  ControlEditTextSelectionRec selectionRec;
  ControlKeyFilterUPP         numericFilterUPP;
  ModalFilterUPP              eventFilterUPP;
  ControlEditTextValidationUPP editTextValidatorUPP;
  SInt16                      tabHit, itemHit;

  if(FrontNonFloatingWindow())
    doActivateWindow(FrontNonFloatingWindow(),false);

  if(!(dialogRef = GetNewDialog(rTabEditClockDialog,NULL,(WindowRef) -1)))
    ExitToShell();

  SetWTitle(GetDialogWindow(dialogRef),"\pTab, Edit Text, and Clock Controls");
  SetPortDialogPort(dialogRef);
  // ............................................... set default button and cursor
tracking

  SetDialogDefaultItem(dialogRef,kStdOkItemIndex);
  SetDialogTracksCursor(dialogRef,true);

  // ................................................. hide user pane with embedded
clocks

  GetDialogItemAsControl(dialogRef,iClocksUserPane,&controlRef);
  HideControl(controlRef);

  // ............... assign some text to the first edit text control and select the whole
text

  GetDialogItemAsControl(dialogRef,iEditText1,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlEditTextTextTag,initialName[0],
                 &initialName[1]);
  selectionRec.selStart = 0;
  selectionRec.selEnd = 32767;
  SetControlData(controlRef,kControlEntireControl,kControlEditTextSelectionTag,
```

```
                sizeof(selectionRec),&selectionRec);

  // create universal procedure pointers for event filter, key filter, and edit text
validator

  eventFilterUPP        = NewModalFilterUPP((ModalFilterProcPtr) eventFilter);
  numericFilterUPP      = NewControlKeyFilterUPP((ControlKeyFilterProcPtr) numericFilter);
  editTextValidatorUPP  = NewControlEditTextValidationUPP((ControlEditTextValidationProcPtr)
                                                          editTextValidator);

  // .................. attach an edit text validation function to the first edit text
control

  GetDialogItemAsControl(dialogRef,iEditText1,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlEditTextValidationProcTag,
                sizeof(editTextValidatorUPP),&editTextValidatorUPP);

// ............................. attach a key filter function to the second edit text
control

  GetDialogItemAsControl(dialogRef,iEditText2,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlEditTextKeyFilterTag,
                sizeof(numericFilterUPP),&numericFilterUPP);

  // ................................ set help tags, show dialog, and enter modal dialog
loop

  if(gRunningOnX)
    helpTagsTabEditClock(dialogRef);

  ShowWindow(GetDialogWindow(dialogRef));

  do
  {
    ModalDialog(eventFilterUPP,&itemHit);

    if(itemHit == iTabs)
    {
      GetDialogItemAsControl(dialogRef,iTabs,&controlRef);
      tabHit = GetControlValue(controlRef);
      if(tabHit == tabEditText)
      {
        SetWTitle(GetDialogWindow(dialogRef),"\pTab, Edit Text, and Clock Controls");

        GetDialogItemAsControl(dialogRef,iClocksUserPane,&controlRef);
        HideControl(controlRef);

        GetDialogItemAsControl(dialogRef,iEditTextUserPane,&controlRef);
        ActivateControl(controlRef);
        ShowControl(controlRef);

        GetDialogItemAsControl(dialogRef,iEditText1,&controlRef);
        SetKeyboardFocus(GetDialogWindow(dialogRef),controlRef,kControlFocusNextPart);
      }
      else if(tabHit == tabClocks)
      {
        SetWTitle(GetDialogWindow(dialogRef),"\pTab, Clock, and Edit Text Controls");

        GetDialogItemAsControl(dialogRef,iEditTextUserPane,&controlRef);
        DeactivateControl(controlRef);
        HideControl(controlRef);

        GetDialogItemAsControl(dialogRef,iClocksUserPane,&controlRef);
        ShowControl(controlRef);

        GetDialogItemAsControl(dialogRef,iClocks1,&controlRef);
```

```
        SetKeyboardFocus(GetDialogWindow(dialogRef),controlRef,kControlFocusNextPart);
      }
    }
    else if(itemHit == iExtractEditText)
    {
      GetDialogItemAsControl(dialogRef,iImageWellEditText,&controlRef);
      Draw1Control(controlRef);
      doExtractEditText(dialogRef);
    }
    else if(itemHit == iExtractClocks)
    {
      GetDialogItemAsControl(dialogRef,iImageWellClocks,&controlRef);
      Draw1Control(controlRef);
      doExtractDateTime(dialogRef);
    }
  }  while(itemHit != kStdOkItemIndex);

  DisposeDialog(dialogRef);
  DisposeModalFilterUPP(eventFilterUPP);
  DisposeControlKeyFilterUPP(numericFilterUPP);
  DisposeControlEditTextValidationUPP(editTextValidatorUPP);
}
// ****************************************************************************
doExtractEditText

void  doExtractEditText(DialogRef dialogRef)
{
  GrafPtr     oldPort;
  RGBColor    saveBackColor, whiteColor = {  0xFFFF, 0xFFFF, 0xFFFF } ;
  SInt16      iType;
  Handle      theHandle;
  Rect        theRect;
  Str255      theString;
  CFStringRef stringRef;
  ControlRef  controlRef;
  Size        actualSize;

  GetPort(&oldPort);
  SetPortDialogPort(dialogRef);
  GetBackColor(&saveBackColor);
  RGBBackColor(&whiteColor);

  GetDialogItem(dialogRef,iEditText1,&iType,&theHandle,&theRect);
  GetDialogItemText(theHandle,theString);
  if(!gRunningOnX)
  {
    MoveTo(124,177);
    DrawString(theString);
  }
  else
  {
    stringRef = CFStringCreateWithPascalString(NULL,theString,kCFStringEncodingMacRoman);
    SetRect(&theRect,124,166,314,181);
    DrawThemeTextBox(stringRef,kThemeSmallSystemFont,true,false,&theRect,teJustLeft,NULL);
  }

  GetDialogItem(dialogRef,iEditText2,&iType,&theHandle,&theRect);
  GetDialogItemText(theHandle,theString);
  if(!gRunningOnX)
  {
    MoveTo(124,190);
    DrawString(theString);
  }
  else
  {
    stringRef = CFStringCreateWithPascalString(NULL,theString,kCFStringEncodingMacRoman);
```

```
    SetRect(&theRect,124,179,314,194);
    DrawThemeTextBox(stringRef,kThemeSmallSystemFont,true,false,&theRect,teJustLeft,NULL);
  }

  GetDialogItemAsControl(dialogRef,iEditText3,&controlRef);
  GetControlDataSize(controlRef,kControlEditTextPart,kControlEditTextTextTag,&actualSize);
  GetControlData(controlRef,kControlEntireControl,kControlEditTextPasswordTag,actualSize,
                 &theString[1],NULL);
  theString[0] = actualSize;
  if(!gRunningOnX)
  {
    MoveTo(124,203);
    DrawString(theString);
  }
  else
  {
    stringRef = CFStringCreateWithPascalString(NULL,theString,kCFStringEncodingMacRoman);
    SetRect(&theRect,124,192,314,207);
    DrawThemeTextBox(stringRef,kThemeSmallSystemFont,true,false,&theRect,teJustLeft,NULL);
  }

  if(gRunningOnX)
  {
    if(stringRef != NULL)
      CFRelease(stringRef);
  }

  RGBBackColor(&saveBackColor);
  SetPort(oldPort);
}

// ****************************************************************************
doExtractDateTime

void  doExtractDateTime(DialogRef dialogRef)
{
  GrafPtr     oldPort;
  RGBColor    saveBackColor, whiteColor = {  0xFFFF, 0xFFFF, 0xFFFF } ;
  ControlRef  controlRef;
  LongDateRec longDateTimeStruc;
  SInt16      second, minute, hour, day, month, year;
  Str255      theString, tempString;
  CFStringRef stringRef;
  Rect        theRect;

  GetPort(&oldPort);
  SetPortDialogPort(dialogRef);
  GetBackColor(&saveBackColor);
  RGBBackColor(&whiteColor);

  GetDialogItemAsControl(dialogRef,iClocks1,&controlRef);
  GetControlData(controlRef,kControlEntireControl,kControlClockLongDateTag,
                 sizeof(longDateTimeStruc),&longDateTimeStruc,NULL);
  hour = longDateTimeStruc.ld.hour;
  minute = longDateTimeStruc.ld.minute;
  second = longDateTimeStruc.ld.second;
  doCopyPString("\pHour ",theString);
  NumToString((SInt32) hour,tempString);
  doConcatPStrings(theString,tempString);
  doConcatPStrings(theString,"\p, Minute ");
  NumToString((SInt32) minute,tempString);
  doConcatPStrings(theString,tempString);
  doConcatPStrings(theString,"\p, Second ");
  NumToString((SInt32) second,tempString);
  doConcatPStrings(theString,tempString);
  MoveTo(124,177);
```

```
  if(!gRunningOnX)
    DrawString(theString);
  else
  {
    stringRef = CFStringCreateWithPascalString(NULL,theString,kCFStringEncodingMacRoman);
    SetRect(&theRect,124,166,314,181);
    DrawThemeTextBox(stringRef,kThemeSmallSystemFont,true,false,&theRect,teJustLeft,NULL);
    if(stringRef != NULL)
      CFRelease(stringRef);
  }

  GetDialogItemAsControl(dialogRef,iClocks2,&controlRef);
  GetControlData(controlRef,kControlEntireControl,kControlClockLongDateTag,
                 sizeof(longDateTimeStruc),&longDateTimeStruc,NULL);
  day = longDateTimeStruc.ld.day;
  month = longDateTimeStruc.ld.month;
  year = longDateTimeStruc.ld.year;
  doCopyPString("\pDay ",theString);
  NumToString((SInt32) day,tempString);
  doConcatPStrings(theString,tempString);
  doConcatPStrings(theString,"\p, Month ");
  NumToString((SInt32) month,tempString);
  doConcatPStrings(theString,tempString);
  doConcatPStrings(theString,"\p, Year ");
  NumToString((SInt32) year,tempString);
  doConcatPStrings(theString,tempString);
  MoveTo(124,190);
  if(!gRunningOnX)
    DrawString(theString);
  else
  {
    stringRef = CFStringCreateWithPascalString(NULL,theString,kCFStringEncodingMacRoman);
    SetRect(&theRect,124,179,314,194);
    DrawThemeTextBox(stringRef,kThemeSmallSystemFont,true,false,&theRect,teJustLeft,NULL);
    if(stringRef != NULL)
      CFRelease(stringRef);
  }

  GetDialogItemAsControl(dialogRef,iClocks3,&controlRef);
  GetControlData(controlRef,kControlEntireControl,kControlClockLongDateTag,
                 sizeof(longDateTimeStruc),&longDateTimeStruc,NULL);
  month = longDateTimeStruc.ld.month;
  year = longDateTimeStruc.ld.year;
  doCopyPString("\pMonth ",theString);
  NumToString((SInt32) month,tempString);
  doConcatPStrings(theString,tempString);
  doConcatPStrings(theString,"\p, Year ");
  NumToString((SInt32) year,tempString);
  doConcatPStrings(theString,tempString);
  MoveTo(124,203);
  if(!gRunningOnX)
    DrawString(theString);
  else
  {
    stringRef = CFStringCreateWithPascalString(NULL,theString,kCFStringEncodingMacRoman);
    SetRect(&theRect,124,192,314,207);
    DrawThemeTextBox(stringRef,kThemeSmallSystemFont,true,false,&theRect,teJustLeft,NULL);
    if(stringRef != NULL)
      CFRelease(stringRef);
  }

  RGBBackColor(&saveBackColor);
  SetPort(oldPort);
}

// ****************************************************************************
```

```
numericFilter

ControlKeyFilterResult  numericFilter(ControlRef controlRef,SInt16* keyCode,SInt16
*charCode,
                                      EventModifiers *modifiers)
{
  if(((char) *charCode >= '0') && ((char) *charCode <= '9') ||
     (BitTst(modifiers,15 - cmdKeyBit)))
  {
    return kControlKeyFilterPassKey;
  }

  switch(*charCode)
  {
    case kLeftArrow:
    case kRightArrow:
    case kUpArrow:
    case kDownArrow:
    case kBackspace:
    case kDelete:
      return kControlKeyFilterPassKey;
      break;
  }

  SysBeep(10);
  return kControlKeyFilterBlockKey;
}

// ************************************************************************
editTextValidator

void  editTextValidator(ControlRef controlRef)
{
  Str255 oldText, newText;
  Size   actualSize;
  UInt8  a, count = 0;

  GetControlData(controlRef,kControlEntireControl,kControlEditTextTextTag,sizeof(oldText) -
1,
                 &oldText[1],&actualSize);

  if(actualSize <= 255)
    oldText[0] = actualSize;
  else
    oldText[0] = 255;
  for(a=1;a<=oldText[0];a++)
  {
    if(((oldText[a] >= 'A') && (oldText[a] <= 'Z') ||
        (oldText[a] >= 'a') && (oldText[a] <= 'z')) ||
        (oldText[a] == ' ') || (oldText[a] == '.'))
    {
      newText[count + 1] = oldText[a];
      count++;
    }
  }

  newText[0] = count;

  SetControlData(controlRef,kControlEntireControl,kControlEditTextTextTag,newText[0],
                 &newText[1]);

  Draw1Control(controlRef);
}

// ************************************************************************
helpTagsTabEditClock
```

```
void  helpTagsTabEditClock(DialogRef dialogRef)
{
  HMHelpContentRec helpContent;
  SInt16           a;
  static SInt16    itemNumber[10] = {  2,5,7,9,11,21,14,16,18,20 } ;
  ControlRef       controlRef;

  memset(&helpContent,0,sizeof(helpContent));
  HMSetTagDelay(500);
  HMSetHelpTagsDisplayed(true);

  helpContent.version = kMacHelpVersion;
  helpContent.tagSide = kHMOutsideTopCenterAligned;
  helpContent.content[kHMMinimumContentIndex].contentType = kHMStringResContent;
  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmResID = 133;

  for(a = 1;a <= 10; a++)
  {
    helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = a;
    GetDialogItemAsControl(dialogRef,itemNumber[a - 1],&controlRef);
    HMSetControlHelpContent(controlRef,&helpContent);
  }
}

//
**********************************************************************************************
// GroupArrowsProgress.c
//
**********************************************************************************************

// ..........................................................................
includes

#include "Controls3.h"
// ..................................................................... global
variables

ControlRef       gCacheSizeControlRef;
ControlRef       gLittleArrowsControlRef;
ControlActionUPP gArrowsActionFunctionUPP;
extern Boolean   gRunningOnX;
extern Boolean   gGroupArrowsProgressActive;

// ********************************************************************
doGroupArrowsProgress

void  doGroupArrowsProgress(void)
{
  DialogRef      dialogRef;
  ControlRef     controlRef;
  ModalFilterUPP eventFilterUPP;
  SInt16         controlValue, itemHit;
  Str255         theString;

  if(FrontNonFloatingWindow())
    doActivateWindow(FrontNonFloatingWindow(),false);

  if(!(dialogRef = GetNewDialog(rGroupArrowsProgDialog,NULL,(WindowRef) -1)))
    ExitToShell();

  SetPortDialogPort(dialogRef);

  // ..................................................... set default button, help
tags
```

```
  SetDialogDefaultItem(dialogRef,kStdOkItemIndex);

  // .. create universal procedure pointers for event filter and little arrows action
function

  eventFilterUPP            = NewModalFilterUPP((ModalFilterProcPtr) eventFilter);
  gArrowsActionFunctionUPP  = NewControlActionUPP((ControlActionProcPtr)
arrowsActionFunction);

  // ..... set initial value for checkbox group
box

  GetDialogItemAsControl(dialogRef,iCheckboxGroup,&controlRef);
  SetControlValue(controlRef,1);

  // ..... get and set initial cache value for little
arrows

  GetDialogItemAsControl(dialogRef,iLittleArrows,&gLittleArrowsControlRef);
  NumToString((SInt32) GetControlValue(gLittleArrowsControlRef),theString);
  doConcatPStrings(theString,"\pK");
  GetDialogItemAsControl(dialogRef,iStaticTextCache,&gCacheSizeControlRef);
  SetControlData(gCacheSizeControlRef,kControlEntireControl,kControlStaticTextTextTag,
                theString[0],&theString[1]);

  // ..... hide second user pane in pop-up group
box

  GetDialogItemAsControl(dialogRef,iUserPaneScoreAverage,&controlRef);
  HideControl(controlRef);
  // ..... set help tags, show dialog, and enter ModalDialog
loop

  if(gRunningOnX)
    helpTagsGroupArrowsProgress(dialogRef);

  ShowWindow(GetDialogWindow(dialogRef));

  do
  {
    ModalDialog(eventFilterUPP,&itemHit);

    if(itemHit == iCheckboxGroup)
    {
      doCheckBoxGroupBox(dialogRef);
    }
    else if(itemHit == iPopupGroup)
    {
      doPopupGroupBox(dialogRef);
    }
    else if(itemHit == iCheckboxShowInitials)
    {
      GetDialogItemAsControl(dialogRef,iCheckboxShowInitials,&controlRef);
      controlValue = (!(GetControlValue(controlRef)));
      SetControlValue(controlRef,controlValue);
    }
    else if(itemHit == iCheckboxShowAverages)
    {
      GetDialogItemAsControl(dialogRef,iCheckboxShowAverages,&controlRef);
      controlValue = (!(GetControlValue(controlRef)));
      SetControlValue(controlRef,controlValue);
    }
    else if(itemHit == iDisclosureTriangle)
    {
      doChasingAndProgress(dialogRef);
    }
```

```
    else if(itemHit == iPushButtonExtract)
    {
      GetDialogItemAsControl(dialogRef,iImageWell,&controlRef);
      Draw1Control(controlRef);
      doExtractCurrentStatus(dialogRef);
    }
  }  while(itemHit != kStdOkItemIndex);

  // ..................................................................... clean
up

  DisposeDialog(dialogRef);
  DisposeModalFilterUPP(eventFilterUPP);
  DisposeControlActionUPP(gArrowsActionFunctionUPP);

  gGroupArrowsProgressActive = false;
}
// *********************************************************************
doCheckBoxGroupBox

void  doCheckBoxGroupBox(DialogRef dialogRef)
{
  ControlRef controlRef;
  SInt16     controlValue;

  GetDialogItemAsControl(dialogRef,iCheckboxGroup,&controlRef);
  controlValue = (!(GetControlValue(controlRef)));
  SetControlValue(controlRef,controlValue);

  if(controlValue == 0)
  {
    GetDialogItemAsControl(dialogRef,iRadioGroupColor,&controlRef);
    DeactivateControl(controlRef);
    GetDialogItemAsControl(dialogRef,iStaticTextColorDepth,&controlRef);
    DeactivateControl(controlRef);
  }
  else if(controlValue == 1)
  {
    GetDialogItemAsControl(dialogRef,iRadioGroupColor,&controlRef);
    ActivateControl(controlRef);
    GetDialogItemAsControl(dialogRef,iStaticTextColorDepth,&controlRef);
    ActivateControl(controlRef);
  }
}

// *********************************************************************
doPopupGroupBox

void  doPopupGroupBox(DialogRef dialogRef)
{
  ControlRef controlRef;
  SInt16     controlValue;

  GetDialogItemAsControl(dialogRef,iPopupGroup,&controlRef);
  controlValue = GetControlValue(controlRef);

  if(controlValue == 1)
  {
    GetDialogItemAsControl(dialogRef,iUserPaneScoreAverage,&controlRef);
    HideControl(controlRef);
    GetDialogItemAsControl(dialogRef,iUserPaneNamesInitials,&controlRef);
    ShowControl(controlRef);
  }
  else if(controlValue == 2)
  {
    GetDialogItemAsControl(dialogRef,iUserPaneNamesInitials,&controlRef);
```

```
    HideControl(controlRef);
    GetDialogItemAsControl(dialogRef,iUserPaneScoreAverage,&controlRef);
    ShowControl(controlRef);
  }
}
// *********************************************************************
doChasingAndProgress

void  doChasingAndProgress(DialogRef dialogRef)
{
  ControlRef controlRef;
  SInt16     controlValue;
  Handle     ditlHdl;
  Boolean    indeterminateFlag = 1;
  Str255     expandString   = "\pHide Progress Bar and Chasing Arrows";
  Str255     collapseString = "\pShow Progress Bar and Chasing Arrows";

  GetDialogItemAsControl(dialogRef,iDisclosureTriangle,&controlRef);
  controlValue = (!(GetControlValue(controlRef)));
  SetControlValue(controlRef,controlValue);

  if(controlValue == 1)
  {
    ditlHdl = GetResource('DITL',130);
    AppendDITL(dialogRef,ditlHdl,appendDITLBottom);
    ReleaseResource(ditlHdl);

    GetDialogItemAsControl(dialogRef,iProgressBar,&controlRef);
    SetControlData(controlRef,kControlEntireControl,kControlProgressBarIndeterminateTag,
                  sizeof(indeterminateFlag),&indeterminateFlag);

    GetDialogItemAsControl(dialogRef,kStdOkItemIndex,&controlRef);
    MoveControl(controlRef,277,351);

    GetDialogItemAsControl(dialogRef,iStaticTextDisclosure,&controlRef);
    SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,
                  expandString[0],&expandString[1]);
    Draw1Control(controlRef);
  }
  else if(controlValue == 0)
  {
    GetDialogItemAsControl(dialogRef,kStdOkItemIndex,&controlRef);
    MoveControl(controlRef,277,280);

    ShortenDITL(dialogRef,3);
    SizeWindow(GetDialogWindow(dialogRef),362,321,false);

    GetDialogItemAsControl(dialogRef,iStaticTextDisclosure,&controlRef);
    SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,
                  collapseString[0],&collapseString[1]);
    Draw1Control(controlRef);
  }
}

// *********************************************************************
doExtractCurrentStatus
void  doExtractCurrentStatus(DialogRef dialogRef)
{
  GrafPtr     oldPort;
  RGBColor    saveBackColor, whiteColor = {  0xFFFF, 0xFFFF, 0xFFFF } ;
  ControlRef  controlRef;
  SInt16      controlValue;
  Str255      theString;
  CFStringRef stringRef;
  Rect        theRect;
```

```
GetPort(&oldPort);
SetPortDialogPort(dialogRef);
GetBackColor(&saveBackColor);
RGBBackColor(&whiteColor);

GetDialogItemAsControl(dialogRef,iCheckboxGroup,&controlRef);
controlValue = GetControlValue(controlRef);
if(controlValue)
{
  doCopyPString("\pUse color,",theString);
  GetDialogItemAsControl(dialogRef,iRadioGroupColor,&controlRef);
  controlValue = GetControlValue(controlRef);
  if(controlValue == 1)
    doConcatPStrings(theString,"\p 8 bit.");
  else if(controlValue == 2)
    doConcatPStrings(theString,"\p 16 bit.");
  else if(controlValue == 3)
    doConcatPStrings(theString,"\p 32 bit.");
}
else
  doCopyPString("\pDont use color.",theString);

if(!gRunningOnX)
{
  MoveTo(108,216);
  DrawString(theString);
}
else
{
  stringRef = CFStringCreateWithPascalString(NULL,theString,kCFStringEncodingMacRoman);
  SetRect(&theRect,108,205,347,220);
  DrawThemeTextBox(stringRef,kThemeSmallSystemFont,true,false,&theRect,teJustLeft,NULL);
  if(stringRef != NULL)
    CFRelease(stringRef);
}

GetDialogItemAsControl(dialogRef,iPopupGroup,&controlRef);
controlValue = GetControlValue(controlRef);
if(controlValue == 1)
{
  doCopyPString("\pPlayer, ",theString);
  GetDialogItemAsControl(dialogRef,iRadioGroupNames,&controlRef);
  controlValue = GetControlValue(controlRef);
  if(controlValue == 1)
    doConcatPStrings(theString,"\pname first,");
  else if(controlValue == 2)
    doConcatPStrings(theString,"\pname last,");
  GetDialogItemAsControl(dialogRef,iCheckboxShowInitials,&controlRef);
  controlValue = GetControlValue(controlRef);
  if(controlValue == 1)
    doConcatPStrings(theString,"\p show number.");
  else if(controlValue == 0)
    doConcatPStrings(theString,"\p no number.");
}
else if(controlValue == 2)
{
  doCopyPString("\pScore, ",theString);
  GetDialogItemAsControl(dialogRef,iRadioGroupScores,&controlRef);
  controlValue = GetControlValue(controlRef);
  if(controlValue == 1)
    doConcatPStrings(theString,"\pbatting, ");
  else if(controlValue == 2)
    doConcatPStrings(theString,"\pbowling, ");
  GetDialogItemAsControl(dialogRef,iCheckboxShowAverages,&controlRef);
  controlValue = GetControlValue(controlRef);
  if(controlValue == 1)
```

```
      doConcatPStrings(theString,"\pshow average.");
    else if(controlValue == 0)
      doConcatPStrings(theString,"\pno average.");
  }

  if(!gRunningOnX)
  {
    MoveTo(108,229);
    DrawString(theString);
  }
  else
  {
    stringRef = CFStringCreateWithPascalString(NULL,theString,kCFStringEncodingMacRoman);
    SetRect(&theRect,108,219,347,234);
    DrawThemeTextBox(stringRef,kThemeSmallSystemFont,true,false,&theRect,teJustLeft,NULL);
    if(stringRef != NULL)
      CFRelease(stringRef);
  }

  if(!gRunningOnX)
  {
    MoveTo(108,242);
    DrawString("\pCache size: ");
  }
  else
  {
    stringRef = CFStringCreateWithPascalString(NULL,"\pCache size: ",
                                               kCFStringEncodingMacRoman);
    SetRect(&theRect,108,232,347,247);
    DrawThemeTextBox(stringRef,kThemeSmallSystemFont,true,false,&theRect,teJustLeft,NULL);
    if(stringRef != NULL)
      CFRelease(stringRef);
  }
  GetDialogItemAsControl(dialogRef,iLittleArrows,&controlRef);
  NumToString((SInt32) GetControlValue(controlRef),theString);
  if(!gRunningOnX)
    DrawString(theString);
  else
  {
    stringRef = CFStringCreateWithPascalString(NULL,theString,kCFStringEncodingMacRoman);
    SetRect(&theRect,174,232,347,247);
    DrawThemeTextBox(stringRef,kThemeSmallSystemFont,true,false,&theRect,teJustLeft,NULL);
    if(stringRef != NULL)
      CFRelease(stringRef);
  }

  RGBBackColor(&saveBackColor);
  SetPort(oldPort);
}

// ************************************************************
helpTagsGroupArrowsProgress

void  helpTagsGroupArrowsProgress(DialogRef dialogRef)
{
  HMHelpContentRec helpContent;
  SInt16           a;
  static SInt16    itemNumber[8] = {  2,9,21,25,27,29,31,32 } ;
  ControlRef       controlRef;

  memset(&helpContent,0,sizeof(helpContent));
  HMSetTagDelay(500);
  HMSetHelpTagsDisplayed(true);

  helpContent.version = kMacHelpVersion;
  helpContent.tagSide = kHMOutsideTopCenterAligned;
```

```
  helpContent.content[kHMMinimumContentIndex].contentType = kHMStringResContent;
  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmResID = 134;

  for(a = 1;a <= 8; a++)
  {
    helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = a;
    GetDialogItemAsControl(dialogRef,itemNumber[a - 1],&controlRef);
    HMSetControlHelpContent(controlRef,&helpContent);
  }
}

//
**********************************************************************************************
// Sliders.c
//
**********************************************************************************************

// ..........................................................................
includes

#include "Controls3.h"
// ..................................................................... global
variables

extern Boolean gRunningOnX;

// ..................................................................... global
variables

ControlActionUPP           gSliderActionFunction1UPP;
ControlActionUPP           gSliderActionFunction2UPP;
ControlUserPaneDrawUPP     gUserPaneDrawFunctionUPP;
ControlUserPaneActivateUPP gUserPaneActivateFunctionUPP;
ControlRef                 gSlider1Ref;
ControlRef                 gSlider2Ref;
ControlRef                 gSlider3Ref;
ControlRef                 gSlider4Ref;
ControlRef                 gSlider5Ref;
ControlRef                 gSlider6Ref;
RGBColor                   gRedColor;
RGBColor                   gBlueColor;
RGBColor                   gBlackColor   = {  0x0000, 0x0000, 0x0000 } ;
Boolean                    gDrawActivated = true;
extern Boolean             gSlidersActive;

// ************************************************************************
doSliderUserPane

void  doSliderUserPane(void)
{
  DialogRef      dialogRef;
  ModalFilterUPP eventFilterUPP;
  ControlRef     controlRef;
  SInt16         itemHit;

  if(FrontNonFloatingWindow())
    doActivateWindow(FrontNonFloatingWindow(),false);

  if(!(dialogRef = GetNewDialog(rSlidersDialog,NULL,(WindowRef) -1)))
    ExitToShell();

  // ..................................................................... set default
button

  SetDialogDefaultItem(dialogRef,kStdOkItemIndex);
```

```
  // ........................... create universal procedure pointer for event filter
function

  eventFilterUPP = NewModalFilterUPP((ModalFilterProcPtr) eventFilter);

  // ......................... create universal procedure pointers for slider action
functions

  gSliderActionFunction1UPP = NewControlActionUPP((ControlActionProcPtr)
                              sliderActionFunction1);
  gSliderActionFunction2UPP = NewControlActionUPP((ControlActionProcPtr)
                              sliderActionFunction2);
  // .... create universal procedure pointers for user pane functions, set user pane
functions

  gUserPaneDrawFunctionUPP = NewControlUserPaneDrawUPP((ControlUserPaneDrawProcPtr)
                                                       userPaneDrawFunction);
  GetDialogItemAsControl(dialogRef,iUserPane1,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlUserPaneDrawProcTag,
                sizeof(gUserPaneDrawFunctionUPP),&gUserPaneDrawFunctionUPP);

  gUserPaneActivateFunctionUPP = NewControlUserPaneActivateUPP
((ControlUserPaneActivateProcPtr)
                                                             userPaneActivateFunction);
  GetDialogItemAsControl(dialogRef,iUserPane1,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlUserPaneActivateProcTag,
                sizeof(gUserPaneActivateFunctionUPP),&gUserPaneActivateFunctionUPP);

  // ........... get control handles of, and draw initial control values for, top four
sliders

  GetDialogItemAsControl(dialogRef,iSlider1,&gSlider1Ref);
  doDrawSliderValues(dialogRef,gSlider1Ref);
  GetDialogItemAsControl(dialogRef,iSlider2,&gSlider2Ref);
  doDrawSliderValues(dialogRef,gSlider2Ref);
  GetDialogItemAsControl(dialogRef,iSlider3,&gSlider3Ref);
  doDrawSliderValues(dialogRef,gSlider3Ref);
  GetDialogItemAsControl(dialogRef,iSlider4,&gSlider4Ref);
  doDrawSliderValues(dialogRef,gSlider4Ref);

  // ........................ get control handles and values for bottom two sliders, set
color

  GetDialogItemAsControl(dialogRef,iSlider5,&gSlider5Ref);
  gRedColor.red = 2 * GetControlValue(gSlider5Ref);
  GetDialogItemAsControl(dialogRef,iSlider6,&gSlider6Ref);
  gBlueColor.blue = 2 * GetControlValue(gSlider5Ref);

  // ................................ set help tags, show dialog, and enter ModalDialog
loop

  if(gRunningOnX)
    helpTagsSliders(dialogRef);

  ShowWindow(GetDialogWindow(dialogRef));

  do
  {
    ModalDialog(eventFilterUPP,&itemHit);
  }  while(itemHit != kStdOkItemIndex);

  // ..................................................................... clean
up

  DisposeDialog(dialogRef);
  DisposeModalFilterUPP(eventFilterUPP);
```

```
  DisposeControlActionUPP(gSliderActionFunction1UPP);
  DisposeControlActionUPP(gSliderActionFunction2UPP);
  DisposeControlUserPaneDrawUPP(gUserPaneDrawFunctionUPP);
  DisposeControlUserPaneActivateUPP(gUserPaneActivateFunctionUPP);
  gSlidersActive = false;
}
// *********************************************************************
doDrawSliderValues

void  doDrawSliderValues(DialogRef dialogRef,ControlRef controlRef)
{
  Str255  theString;
  SInt16  staticTextItem;

  NumToString((SInt32) GetControlValue(controlRef),theString);

  if(controlRef == gSlider1Ref)
    staticTextItem = iSlider1StaticText;
  else if(controlRef == gSlider2Ref)
    staticTextItem = iSlider2StaticText;
  else if(controlRef == gSlider3Ref)
    staticTextItem = iSlider3StaticText;
  else if(controlRef == gSlider4Ref)
    staticTextItem = iSlider4StaticText;

  GetDialogItemAsControl(dialogRef,staticTextItem,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,theString[0],
                 &theString[1]);
  Draw1Control(controlRef);
}

// *********************************************************************
userPaneDrawFunction

void  userPaneDrawFunction(ControlRef theControl,SInt16 thePart)
{
  Rect theRect;

  SetRect(&theRect,218,175,238,195);
  DrawThemePlacard(&theRect,gDrawActivated);
  InsetRect(&theRect,2,2);

  if(gDrawActivated)
  {
    RGBForeColor(&gRedColor);
    PaintRect(&theRect);
  }

  SetRect(&theRect,218,196,238,216);
  DrawThemePlacard(&theRect,gDrawActivated);
  InsetRect(&theRect,2,2);

  if(gDrawActivated)
  {
    RGBForeColor(&gBlueColor);
    PaintRect(&theRect);
  }

 }
// *********************************************************************
userPaneActivateFunction

void  userPaneActivateFunction(ControlRef control,Boolean activating)
{
  if(activating)
    gDrawActivated = true;
```

```
  else
    gDrawActivated = false;
}

// ************************************************************************
helpTagsSliders

void  helpTagsSliders(DialogRef dialogRef)
{
  HMHelpContentRec helpContent;
  SInt16           a;
  static SInt16    itemNumber[7] = {  2,3,4,5,17,19,18 } ;
  ControlRef       controlRef;

  memset(&helpContent,0,sizeof(helpContent));
  HMSetTagDelay(500);
  HMSetHelpTagsDisplayed(true);

  helpContent.version = kMacHelpVersion;
  helpContent.tagSide = kHMOutsideTopCenterAligned;
  helpContent.content[kHMMinimumContentIndex].contentType = kHMStringResContent;
  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmResID = 135;

  for(a = 1;a <= 7; a++)
  {
    helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = a;
    GetDialogItemAsControl(dialogRef,itemNumber[a - 1],&controlRef);
    HMSetControlHelpContent(controlRef,&helpContent);
  }
}

//
****************************************************************************************
// TextBox.c
//
****************************************************************************************

// ..........................................................................
includes

#include "Controls3.h"

// ************************************************************************
doTextBox

void  doTextBox(void)
{
  DialogRef      dialogRef;
  ModalFilterUPP eventFilterUPP;
  SInt16         itemHit;

  if(FrontNonFloatingWindow())
    doActivateWindow(FrontNonFloatingWindow(),false);
  if(!(dialogRef = GetNewDialog(rAboutDialog,NULL,(WindowRef) -1)))
    ExitToShell();

  // ..................................................................... set default
button

  SetDialogDefaultItem(dialogRef,kStdOkItemIndex);

  // ................................... create universal procedure pointer for event
filter

  eventFilterUPP = NewModalFilterUPP((ModalFilterProcPtr) eventFilter);
```

```
  // ..... show dialog and enter modal dialog
loop

  ShowWindow(GetDialogWindow(dialogRef));

  do
  {
    ModalDialog(eventFilterUPP,&itemHit);
  }  while(itemHit != kStdOkItemIndex);

  DisposeDialog(dialogRef);
  DisposeModalFilterUPP(eventFilterUPP);
}

//
**********************************************************************************************
// Callbacks.c
//
**********************************************************************************************

// ..........................................................................
includes

#include "Controls3.h"

// ..... global
variables

extern Boolean          gGroupArrowsProgressActive;
extern Boolean          gSlidersActive;
extern ControlActionUPP gArrowsActionFunctionUPP;
extern ControlActionUPP gSliderActionFunction1UPP;
extern ControlActionUPP gSliderActionFunction2UPP;
extern ControlRef       gLittleArrowsControlRef;
extern ControlRef       gCacheSizeControlRef;
extern ControlRef       gSlider1Ref;
extern ControlRef       gSlider2Ref;
extern ControlRef       gSlider3Ref;
extern ControlRef       gSlider4Ref;
extern ControlRef       gSlider5Ref;
extern ControlRef       gSlider6Ref;
extern RGBColor         gRedColor;
extern RGBColor         gBlueColor;
extern RGBColor         gBlackColor;
// ************************************************************************
eventFilter

Boolean  eventFilter(DialogRef dialogRef,EventRecord *eventStrucPtr,SInt16 *itemHit)
{
  Boolean    handledEvent;
  GrafPtr    oldPort;
  Point      mouseXY;
  ControlRef controlRef;

  handledEvent = false;

  if((eventStrucPtr->what == updateEvt) &&
     ((WindowRef) eventStrucPtr->message != GetDialogWindow(dialogRef)))
  {
    doUpdate(eventStrucPtr);
  }
  else if((eventStrucPtr->what == autoKey) && ((eventStrucPtr->modifiers & cmdKey) != 0))
  {
    handledEvent = true;
    return handledEvent;
  }
```

```
  else
  {
    GetPort(&oldPort);
    SetPortDialogPort(dialogRef);

    if(gGroupArrowsProgressActive)
    {
      if(eventStrucPtr->what == mouseDown)
      {
        mouseXY = eventStrucPtr->where;
        GlobalToLocal(&mouseXY);
        if(FindControl(mouseXY,GetDialogWindow(dialogRef),&controlRef))
        {
          if(controlRef == gLittleArrowsControlRef)
          {
            TrackControl(controlRef,mouseXY,gArrowsActionFunctionUPP);
            handledEvent = true;
          }
        }
      }
    }
    else if(gSlidersActive)
    {
      if(eventStrucPtr->what == mouseDown)
      {
        mouseXY = eventStrucPtr->where;
        GlobalToLocal(&mouseXY);
        if(FindControl(mouseXY,GetDialogWindow(dialogRef),&controlRef))
        {
          if(controlRef == gSlider1Ref || controlRef == gSlider2Ref)
          {
            TrackControl(controlRef,mouseXY,NULL);
            doDrawSliderValues(dialogRef,controlRef);
            handledEvent = true;
          }
          else if(controlRef == gSlider3Ref || controlRef == gSlider4Ref)
          {
            TrackControl(controlRef,mouseXY,gSliderActionFunction1UPP);
            handledEvent = true;
          }
          else if(controlRef == gSlider5Ref || controlRef == gSlider6Ref)
          {
            TrackControl(controlRef,mouseXY,gSliderActionFunction2UPP);
            handledEvent = true;
          }
        }
      }
    }
    else
    {
      handledEvent = StdFilterProc(dialogRef,eventStrucPtr,itemHit);
    }

    SetPort(oldPort);
  }

  return handledEvent;
}

// *********************************************************************
arrowsActionFunction

void  arrowsActionFunction(ControlRef controlRef,SInt16 partCode)
{
  Str255 theString;
  SInt32 controlValue;
```

```
  if(partCode)
  {
    controlValue = GetControlValue(controlRef);

    switch(partCode)
    {
      case kControlUpButtonPart:
        controlValue += 32;
        if(controlValue > GetControlMaximum(controlRef))
        {
          controlValue = GetControlMaximum(controlRef);
          return;
        }
      break;

      case kControlDownButtonPart:
        controlValue -= 32;
        if(controlValue < GetControlMinimum(controlRef))
        {
          controlValue = GetControlMinimum(controlRef);
          return;
        }
        break;
    }
    SetControlValue(controlRef,controlValue);

    NumToString((SInt32) controlValue,theString);
    doConcatPStrings(theString,"\pK");
    SetControlData(gCacheSizeControlRef,kControlEntireControl,kControlStaticTextTextTag,
                   theString[0],&theString[1]);
    Draw1Control(gCacheSizeControlRef);
  }
}

// ********************************************************************
sliderActionFunction1

void  sliderActionFunction1(ControlRef controlRef,SInt16 partCode)
{
  SInt16    staticTextItem;
  Str255    theString;
  DialogRef dialogRef;

  NumToString((SInt32) GetControlValue(controlRef),theString);

  dialogRef = GetDialogFromWindow(GetControlOwner(controlRef));

  if(controlRef == gSlider3Ref)
    staticTextItem = iSlider3StaticText;
  else if(controlRef == gSlider4Ref)
    staticTextItem = iSlider4StaticText;

  GetDialogItemAsControl(dialogRef,staticTextItem,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,theString[0],
                 &theString[1]);
  Draw1Control(controlRef);
}

// ********************************************************************
sliderActionFunction2

void  sliderActionFunction2(ControlRef controlRef,SInt16 partCode)
{
  SInt16 controlValue;
  Rect   theRect;
```

```
  controlValue = GetControlValue(controlRef);

  if(controlRef == gSlider5Ref)
  {
    gRedColor.red = 2 * controlValue;
    RGBForeColor(&gRedColor);
    SetRect(&theRect,220,177,236,193);
  }
  else if(controlRef == gSlider6Ref)
  {
    gBlueColor.blue = 2 * controlValue;
    RGBForeColor(&gBlueColor);
    SetRect(&theRect,220,198,236,214);
  }
  PaintRect(&theRect);
  RGBForeColor(&gBlackColor);
}

//
********************************************************************************************
// SmallControls.c
//
********************************************************************************************

// ..........................................................................
includes

#include "Controls3.h"

// ***************************************************************************
doSmallControls

void  doSmallControls(void)
{
  OSStatus            osError;
  Str255              pascalString;
  CFStringRef         stringRef;
  Rect                contentRect        = {    0,  0,388,175 } ;
  Rect                staticTextRect     = {    8, 20, 86,144 } ;
  Rect                vertScrollBarRect  = {   -1,164,378,175 } ;
  Rect                horizScrollBarRect = {  377, -1,388,165 } ;
  Rect                tabRect            = {  106, 20,144,144 } ;
  Rect                sliderRect         = {  164, 20,182,145 } ;
  Rect                radioButtonRect    = {  200, 23,216,144 } ;
  Rect                checkBoxRect       = {  220, 23,236,144 } ;
  Rect                editTextRect       = {  300, 26,313,139 } ;
  Rect                pushButtonRect     = {  338, 21,356,144 } ;
  WindowRef           windowRef;
  ControlFontStyleRec controlFontStyleStruc;
  ControlRef          controlRef;
  ControlSize         controlSize = kControlSizeSmall;
  ControlTabEntry     tabArray[2];

  if(!(osError = CreateNewWindow(kFloatingWindowClass,kWindowStandardFloatingAttributes +
                                 kWindowResizableAttribute,&contentRect,
                                 &windowRef)))
  {
    ChangeWindowAttributes(windowRef,0,kWindowCloseBoxAttribute);
    RepositionWindow(windowRef,NULL,kWindowAlertPositionOnMainScreen);
    SetThemeWindowBackground(windowRef,kThemeBrushDialogBackgroundActive,false);
    SetPortWindowPort(windowRef);
    UseThemeFont(kThemeSmallSystemFont,smSystemScript);
    ShowWindow(windowRef);
  }
  else
```

```
  {
    SysBeep(10);
    return;
  }

  GetIndString(pascalString,rSmallControlsString,1);
  stringRef = CFStringCreateWithPascalString(NULL,pascalString,kCFStringEncodingMacRoman);
  controlFontStyleStruc.flags = kControlUseFontMask | kControlUseJustMask;
  controlFontStyleStruc.font   = kControlFontSmallSystemFont;
  controlFontStyleStruc.just  = teJustCenter;

  CreateStaticTextControl(windowRef,&staticTextRect,stringRef,&controlFontStyleStruc,
                          &controlRef);

  // ..................................................................... scroll
bars

  if((osError = CreateScrollBarControl(windowRef,&vertScrollBarRect,0,0,200,200,false,NULL,
                                       &controlRef)) == noErr)
    SetControlData(controlRef,kControlEntireControl,kControlSizeTag,sizeof(controlSize),
                   &controlSize);

  if((osError = CreateScrollBarControl(windowRef,&horizScrollBarRect,0,0,300,300,false,NULL,
                                       &controlRef)) == noErr)
    SetControlData(controlRef,kControlEntireControl,kControlSizeTag,sizeof(controlSize),
                   &controlSize);

  // ..................................................................................
tab

  tabArray[0].icon = tabArray[1].icon = NULL;
  tabArray[0].name = CFSTR("Tab 1");
  tabArray[1].name = CFSTR("Tab 2");
  tabArray[0].enabled = tabArray[1].enabled = true;

  CreateTabsControl(windowRef,&tabRect,kControlTabSizeSmall,kControlTabDirectionSouth,2,
                    tabArray,&controlRef);

  // ..................................................................................
slider

  if((osError = CreateSliderControl(windowRef,&sliderRect,0,0,100,
                                    kControlSliderPointsDownOrRight,5,false,NULL,
                                    &controlRef)) == noErr)
    SetControlData(controlRef,kControlEntireControl,kControlSizeTag,sizeof(controlSize),
                   &controlSize);

  // ..................................................................... radio
button

  if((osError = CreateRadioButtonControl(windowRef,&radioButtonRect,
                                         CFSTR("Small Radio Button"),1,false,
                                         &controlRef)) == noErr)
  {
    SetControlData(controlRef,kControlEntireControl,kControlSizeTag,sizeof(controlSize),
                   &controlSize);
    SetControlFontStyle(controlRef,&controlFontStyleStruc);
  }

  // ..................................................................................
checkbox

  if((osError = CreateCheckBoxControl(windowRef,&checkBoxRect,CFSTR("Small Checkbox"),1,
false,
                                      &controlRef)) == noErr)
  {
```

```
    SetControlData(controlRef,kControlEntireControl,kControlSizeTag,sizeof(controlSize),
                   &controlSize);
    SetControlFontStyle(controlRef,&controlFontStyleStruc);
  }

  // ..................................................................... pop-up menu
button

  if(controlRef = GetNewControl(cSmallPopup,windowRef))
    SetControlValue(controlRef,3);

  // ..................................................................... edit
text

  if((osError = CreateEditTextControl(windowRef,&editTextRect,CFSTR("Small Edit Text"),
false,
                                      false,&controlFontStyleStruc,&controlRef)) == noErr)
    SetKeyboardFocus(windowRef,controlRef,kControlFocusNextPart);

  // ..................................................................... push
button

  if((osError = CreatePushButtonControl(windowRef,&pushButtonRect,CFSTR("Small Push
Button"),
                                        &controlRef)) == noErr)
    SetControlFontStyle(controlRef,&controlFontStyleStruc);
}

// ************************************************************
doMouseDownSmallControls

void  doMouseDownSmallControls(WindowRef windowRef,EventRecord *eventStrucPtr)
{
  SInt16      controlPartCode;
  ControlRef  controlRef = NULL;
  ControlKind controlKind;

  SetPortWindowPort(windowRef);
  GlobalToLocal(&eventStrucPtr->where);
  controlPartCode = FindControl(eventStrucPtr->where,windowRef,&controlRef);
  if(controlPartCode)
    TrackControl(controlRef,eventStrucPtr->where,NULL);

  if(controlRef)
  {
    GetControlKind(controlRef,&controlKind);
    if((controlKind.kind == kControlKindRadioButton) ||
        controlKind.kind == kControlKindCheckBox)
      SetControlValue(controlRef,!GetControlValue(controlRef));
  }
}

//
*****************************************************************************************
```

Demonstration Program Controls3 Comments

When this program is run, the user should

- Choose items from the Demonstration menu to view and operate the various controls. (Note that the Small Controls item is only available when the program is run on Mac OS X.)
- On Mac OS 8/9, choose Show Balloons from the Help menu and note the information in the help balloons as the cursor is moved over the various controls.
- On Mac OS X, hover the cursor over the various controls and note the information in the help tags.
- Click in the Finder and then back in the window/dialog, noting control activation/deactivation.
- In the Bevel Buttons, Image Wells, Picture Controls, and Icon Controls window
    - Click in the various controls, noting in the window header the control part code returned by FindControl (or immediately by TrackControl in the case of the non-tracking icon controls).
    - Click in the various controls and then release the mouse button both within and outside the control, noting the control part code returned by TrackControl.
- In the Tab, Edit Text, and Clock Controls dialog
    - In the Edit Text Controls tab
        - Change the keyboard focus using the tab key and mouse clicks.
        - Enter a name, age, and password, noting the effect of the key filter function attached to the "Age" edit text control and the behavior of the "Password" edit text control.
        - Paste some text containing characters other than alphabetic characters, the space character, and the period character to the "Name" edit text control, noting that the edit text validation function strips out the "illegal" characters.
        - Note the cursor shape changes when the cursor is over the top two edit text controls.
        - Click the Extract push button to extract and display the contents of the edit text controls.
    - In the Clock Controls Tab
        - Change the keyboard focus using the tab key and mouse clicks.
        - Change the clock setting by typing and by using the integral little arrows.
        - Click the Extract push button to extract and display the clock settings.
- In the Group Boxes, Arrows, and Progress Bar dialog
    - Change the group box settings, the settings of the controls within the group boxes, and the (simulated) cache size setting controlled by the little arrows, and then click the Extract button to extract and display the settings of the various controls.
    - Click the disclosure triangle to show and hide the chasing arrows and indeterminate progress bar.
- In the Sliders and User Pane Functions dialog
    - Operate the sliders and note the difference in the appearance of the dragged indicator in the live scrolling and non-live scrolling variants of the sliders.
    - Note the appearance, in the activated and deactivated modes, of the two custom "controls" to the right of the two horizontal sliders. Also note that these controls are redrawn, in the appropriate mode, when an overlaying window or help balloon is removed (Mac OS 8/9).
- In the Text Boxes dialog, observe the auto scrolling variant and scroll the non-auto-scrolling variant.

- On Mac OS X, observe the small controls in the Small Controls floating window.

Control3.h

defines

Constants are established for the resource IDs and dialog item numbers associated with the various demonstrations.

typedefs

A variable of type `BevelDocStruc` will be used to hold references to the various controls created in the Bevel Buttons, Image Wells, Picture Controls, and Icon Controls window.

Control3.c

`Controls3.c` is simply the basic "engine" which supports the demonstration. There is nothing in this file which has not featured in previous demonstration programs.

doMouseDown

In the `inContent` case, if a mouse-down occurs in the Bevel Buttons, Image Wells, Picture Controls, and Icon Controls window when it is the front window, the function `doBevelImagePictIconContent` is called.

In the `inGoAway` case, the Bevel Buttons, Image Wells, Picture Controls, and Icon Controls window is disposed of.

doUpdate

If the Bevel Buttons, Image Wells, Picture Controls, and Icon Controls window is open, `UpdateControls` is called to redraw the controls in the appropriate mode.

doActivateWindow

If the Bevel Buttons, Image Wells, Picture Controls, and Icon Controls window is open, and if it receives an activate event, `GetRootControl` is called to get a reference to the window's root control. The controls are then activated or deactivated en masse.

BevelImagePictIcon.c

`BevelImagePictIcon.c` contains most of the source code relating to the Bevel Buttons, Image Wells, Picture Controls, and Icon Controls window.

doBevelImagePictIcon

`doBevelImagePictIcon` opens a window, creates a relocatable block for a structure of type `BevelDocStruc` (to whose fields references to the various controls will be assigned) and stores the handle to this block in the window object.

The call to `GetNewCWindow` then creates the window. (Note that error handling here and in other areas of the demonstration is somewhat rudimentary in that the program simply terminates.) `NewHandle` creates the block for the variable of type `BevelDocStruc` and `SetWRefCon` stores the handle to the block as a reference constant in the window object.

Before `ShowWindow` is called to make the window visible, the function `doGetControls` is called to create the controls.

doGetControls

`doGetControls` creates the controls from the various `'CNTL'` resources.

At the first line, if the program is running on Mac OS 8/9, `CreateRootControl` is called to create a root control for the window. On Mac OS 8/9, the first control created must be always be the root control (which is implemented as a user pane). This call is not necessary on Mac OS X because, on Mac OS X, root controls are created automatically for windows which have at least one control.

A handle to the structure in which the reference to the control objects will be stored is then retrieved. The following calls to `GetNewControl` create a control object for each control, insert the reference to the object into the control list for the specified window and draw the control. At the same time, the reference to each control object is assigned to the appropriate field of the window's "document" structure.

At the next block, if the program is running on Mac OS X, `SetControlData` is called to make the first four bevel button's corners rounded. (Mac OS X bevel buttons have square corners by default.)

At the next block, `SetControlData` is called twice to cause the text in the specified bevel buttons to be placed above the graphic.

At the next block, `SetControlData` is called to cause the pop-up glyph in the specified bevel button to be centered.

At the next block, the font for the text in the specified bevel button is changed. The flags and font fields of a control font style structure are assigned constants so that the following call to `SetControlFontStyle` will set the font to the small emphasized system font.

The final line sets the value of the specified bevel button to 2, causing it to appear in the mixed state.

doBevelImagePictIconContent

Recall that if a mouse-down occurs in the Bevel Buttons, Image Wells, Picture Controls, and Icon Controls window when it is the front window, `doBevelImagePictIconContent` is called to further handle the event.

At the first line a handle to the structure containing references to all the control objects is obtained.

The call to `GlobalToLocal` changes the coordinates at which the mouse-down occurred to the local coordinates required by the following call to `FindControl`. The constant representing the part code returned by `FindControl` is then drawn in the window header.

If the part code returned by `FindControl` is not `kControlNoPart` (0), meaning that an enabled control was under the mouse cursor at the time of the mouse-down, the if block executes. `TrackControl` takes control while the mouse button remains down, returning a part code when the button is released.

In the case of the first 10 specified controls (bevel buttons), the control part code returned by `TrackControl` is simply drawn in the window header.

The next four controls are bevel buttons with menus. In these cases, if the part code returned indicates that the mouse button was released while the cursor was within the menu, `GetBevelButtonMenuValue` is called to get the menu item number. The part code and menu item number are then drawn in the window header. If the cursor was not in the menu when the mouse button was released, the part code returned by `TrackControl` is drawn in the window header.

The 10th, 11th, and 12th bevel buttons have toggle behavior. With some assistance from the application, they behave like radio buttons. If the control in which the mouse-down occurred was one of these bevel buttons, and if the cursor was still within the control when the mouse button was released, `SetControlValue` is called to set the three controls to the unchecked state, following which the selected control is set to the checked state. Either way, the part code returned by `TrackControl` is drawn in the window header.

The 18th bevel button is used to demonstrate bevel button graphic alignments using a single bevel button. A mouse-down within this button causes the function `doGraphicAlignment` to be called.

The 19th bevel button is used to demonstrate bevel button text alignments using a single bevel button. A mouse-down within this button causes the function `doTextAlignment` to be called.

The 20th bevel button is used to demonstrate text offsetting using a single bevel button. A mouse-down within this button causes the function `doTextOffset` to be called.

The 21st bevel button is used to demonstrate bevel button text placement in relation to the bevel button's graphic using a single bevel button. A mouse-down within this button causes the function `doTextPlacement` to be called.

Finally, if the mouse-down was in an image well, picture control, or icon control, and except for the non-tracking picture and icon control variants, the part code returned by `TrackControl` is drawn in the window header.

doDrawPartCode

doDrawPartCode takes part codes and menu item numbers and assembles them into descriptive strings for drawing in the window header. The constants used are stored in 'STR#' resources, and are retrieved using GetIndString. In the final two lines, Draw1Control is called to redraw the window header control (in effect erasing the current text drawn within the window header area) and doDrawMessage is called to draw the assembled string.

doGraphicAlignment

doGraphicAlignment is called to demonstrate bevel button graphic alignment using the helper function SetBevelButtonGraphicAlignment, which facilitates finer adjustment of graphic placement than using SetControlData with the kControlBevelButtonGraphicAlignTag control data tag constant. Each time around the outer for loop, the alignment constant is changed. In the inner for loop, SetBevelButtonGraphicAlignment is called 53 times, with the two offset parameters incremented and Draw1Control called at each call. The alignment constant used during each pass through the outer for loop is drawn in the window header.

doTextAlignment

doTextAlignment is called to demonstrate the effect of the bevel button text alignment constants using the helper function SetBevelButtonTextAlignment, which facilitates finer adjustment of text placement that using SetControlData with the kControlBevelButtonGraphicAlignTag control data tag constant. Each time around the outer for loop, the alignment constant is changed. In the inner for loop, SetBevelButtonTextAlignment is called 41 times, with the offset parameter incremented and Draw1Control called at each call. The alignment constant used during each pass through the outer for loop is drawn in the window header.

doTextOffset

doTextOffset is called to demonstrate text offsetting from the left and the right within a bevel button. For the purposes of the demonstration, two animations involving incrementing offsets values are used. Prior to first animation, SetControlData is called with the kControlBevelButtonTextOffsetTag tag to align the text on the left. Prior to second animation, SetControlData is called with the kControlBevelButtonTextOffsetTag tag to align the text on the right. Within the for loops, SetControlData is called with the kControlBevelButtonTextOffsetTag tag to offset the text from the left or right by the specified number of pixels, and Draw1Control redraws the control.

doTextPlacement

doTextPlacement is called to demonstrate the effect of the bevel button text placement constants. Within a for loop which increments the text placement constant, SetBevelButtonTextPlacement is called and Draw1Control redraws the control.

doDrawMessage and doDrawLegends

doDrawMessage and doDrawLegends are incidental to the demonstration. Both functions draw text in the window in either black or gray depending on whether the window is currently in front or in the background.

TabEditClock.c

TabEditClock.c creates a movable modal dialog in which is demonstrated a tab control, edit text controls and clock controls. A numeric key filter function is attached to the second edit text control. The third edit text control is for password input. The controls displayed by each tab are embedded in user panes.

The kDialogFlagsUsesControlHierarchy flag is set in the 'dlgx' resources for all dialogs used in the demonstration. Recall that this means that the Dialog Manager creates a root control in the dialog and establishes an embedding hierarchy, that all dialog items automatically become controls, and that the Dialog Manager uses AutoEmbedControl to position dialog items in an embedding hierarchy based on both visual containment and their item list number.

The dialog item list used by the dialog created by TabEditClock.c is shown in the following, in which the indentation represents the embedding hierarchy:

```
1.   OK push button primitive.
2.   Tab control (kTabControlLargeProc variant).
      Visually contains the two user panes (items 3 and 12) and thus automatically embeds
those items.
```

```
      3.   User pane control with kControlSupportsEmbedding feature bit set (initial value
set to 2).
            Visually contains items 4 to 11 and thus automatically embeds those items.
            4.   Static text primitive "Name:".
            5     Edit text primitive.
            6.   Static text primitive "Age:".
            7.   Edit text primitive.
            8.   Static text primitive "Password:".
            9.   Edit text control (kControlEditTextPasswordProc variant).
            10.  Extract push button primitive.
            11.  Image well control.
     12.   User pane control with kControlSupportsEmbedding feature bit set (initial value
set to 2).
            Visually contains items 13 to 20 and thus automatically embeds those items.
            13.   Static text primitive "Hours, Mins, Secs:".
            14.   Clock control (kControlClockTimeSecondsProc variant).
            15.   Static text primitive "Date, Month, Year:".
            16.   Clock control (kControlClockDateProc variant).
            17.   Static text primitive "Month, Year:".
            18.   Clock control (kControlClockMonthYearProc variant).
            19.   Extract push button primitive.
            20.   Image well control.
21.   Clock control (kControlClockTimeSecondsProc variant, live, display only.)
22.   Group box (kControlGroupBoxSecondaryTextTitleProc variant).
       Visually contains item 23 and thus automatically embeds that item:
       23.   Static text primitive.
```

doTabEditClock

`doTabEditClock` creates the dialog, calls `ModalDialog` to handle events in the dialog, and disposes of the dialog when the user hits the OK push button.

As is always required when a dialog is to be displayed, the front window (if one exists) is explicitly deactivated.

The call to `GetNewDialog` creates the dialog. The call to `SetWTitle` sets the window's title so as to force an association with the `'hrct'` resource containing the help balloons for the Edit Text Control tab. (Later calls to `SetWTitle` in this function are used for similar purposes.)

`SetDialogDefaultItem` tells the Dialog Manager which is the default push button item, to alias the Return and Enter keys to that item, and to draw the default ring around that item (Mac OS 8/9) or make it pulsing blue (Mac OS X). `SetDialogTracksCursor` will cause the Dialog Manager to change the cursor shape to the I-Beam cursor whenever the cursor is over an edit text item specified in the `'DITL'` as an edit text primitive.

The first call to `GetDialogItemAsControl` gets a reference to the user pane used for the clocks tab. `HideControls` hides this user pane and all the controls automatically embedded within it.

At the next block, `SetControlData` is called with the `kControlEditTextTextTag` tag to assign some text to the first edit text control. The fields of an edit text selection structure are then assigned values which will cause the following call to `SetControlData` to select the entire edit text control.

The next block creates universal procedure pointers for event filter, key filter, and edit text validator callback functions. (The event filter function is defined in the source code file `Callbacks.c`.) `SetControlData` is then called with the `kControlEditTextValidationProcTag` tag to attach the edit text validation function to the first edit text control. `SetControlData` is then called again with the `kControlEditTextKeyFilterTag` tag to attach the key filter function to the second edit text control.

With those preliminaries attended to, `ShowWindow` is called to display the window, following which the `ModalDialog` loop is entered. The loop will execute until the user hits the OK push button. Note that the previously created UPP is passed in the `filterProc` parameter of `ModalDialog`.

When a mouse-down event occurs in an enabled item, `ModalDialog` returns the item number of the item hit. If the item hit was the tab item, `GetDialogItemAsControl` is called to get a reference to the tab control and `GetControlValue` is called to determine which of the two tabs was hit. If the tab hit was the Edit Text Controls tab, the Clock Controls user pane is hidden, the Edit

Text Controls user pane is shown, and the keyboard focus is set to the first edit text control. If the tab hit was the Clock Controls tab, the Edit Text Controls user pane is hidden, the Clock Controls user pane is shown, and the keyboard focus is set to the first clock control.

If the item hit was the Extract push button in the Edit Text Controls pane, the image well control is redrawn to erase any previous text and a function is called to extract and display the contents of the edit text controls.

If the item hit was the Extract push button in the Clock Controls pane, the image well control is redrawn to erase any previous text and a function is called to extract and display the contents of the clock controls.

When the user hits the OK button, `DisposeDialog` closes the dialog and disposes of the associated memory, and the universal procedure pointers associated with the event filter function and key filter function are disposed of.

doExtractEditText

`doExtractEditText` extracts the text from the edit text controls and draws it in the image well area. In the case of the Password edit text control, the text extracted is the actual password typed, not the bullet text.

In the case of the first two edit text controls, `GetDialogItem` is used to get the handle to the `hText` field of the `TERec` structure used by the item. This is where the characters are located. (Exactly what is returned in the `iHandle` field of a `GetDialogItem` call depends on the item type.) `GetDialogItemText` then copies these characters to a variable of type `Str255`.

In the case of the third (password) edit text control, `GetDialogItemAsControl` gets a reference to the control. `GetControlDataSize` is then called with the `kControlEditTextTextTag` tag to get the number of characters and `GetControlData` is called with the `kControlEditTextPasswordTag` to get the clear password text from the control.

doExtractDateTime

`doExtractDateTime` gets the settings in the three clock controls into a long date structure, extracts the content of the relevant fields of that structure, and draws that content in the image well area.

For each clock, `GetControlData` is called with the `kControlClockLongDateTag` tag to get the information into a long date structure. The rest of the code simply converts the values in the relevant fields of this structure to strings, which are concatenated with other identifying strings prior to being drawn in the image well.

numericFilter

`numericFilter` is the key filter function attached to the second edit text control.

The character code and modifiers parameters passed to this function by the control are first examined to determine whether the character code represents the numeric characters between 0 and 9 inclusive or whether the Command key was down. If either of these conditions is true, the function returns `kControlKeyFilterPassKey`. This means that, if the character is between 0 and 9 inclusive, it will be accepted. It also means that, if the Command key was down, non-numeric characters will be accepted as well. This latter is necessary to ensure that the user can edit the text in the edit text control using the usual Edit menu Command-key equivalents.

If a return does not occur at that point, the character code is further examined to determine whether it represents one of the arrow keys, the backspace key or the delete key. If so, `kControlKeyFilterPassKey` is returned. If not, the system alert sound is played and `kControlKeyFilterBlockKey` is returned, meaning that the character is to be rejected.

editTextValidator

`editTextValidator` is the edit text validation callback function attached to the first edit text control.

The call to `GetControlData` gets the text to be examined from the control into the variable `oldText`. The next block sets the length byte in `oldText` to the actual number of characters in the text, limited to 255 characters. The for loop then examines each character and, if it is an alphabetic character, the space character, or the period character, inserts it into the `newText` variable and increments a variable which is eventually used to set the length byte of `newText`. When the if loop exits, the length byte is set and `SetControlData` is called to put the replacement text into the control. Finally, `Draw1Control` redraws the control.

GroupArrowsProgress.c

GroupArrowsProgress.c creates a movable modal dialog in which is demonstrated a checkbox group box, a pop-up group box, little arrows, a disclosure triangle, chasing arrows, and an indeterminate progress bar.

The checkbox group box contains a static text item and three radio buttons. The radio buttons are embedded in a radio group. The pop-up group box contains two user panes. Embedded in each user pane are a checkbox and a radio group. Embedded in each radio group are two radio buttons.

The dialog item list used by the dialog created by GroupArrowsProgress.c is shown in the following, in which the indentation represents the embedding hierarchy:

```
1.   Push button primitive.
2.   Group box control (kControlGroupBoxCheckboxProc variant).
      3.   Radio group control.
            4.   Radio button primitive.
            5.   Radio button primitive.
            6.   Radio button primitive.
      7.   Checkbox primitive.
8.   Group box control(kControlGroupBoxPopupButtonProc variant).
      9.   User pane control with kControlSupportsEmbedding feature bit set (initial value
set to 2).
            10.   Radio group control
                   11.   Radio button primitive.
                   12.   Radio button primitive.
            13.   Checkbox primitive.
            14.   Static text primitive.
      15.  User pane control with kControlSupportsEmbedding feature bit set (initial value
set to 2).
            16.   Radio group control
                   17.   Radio button primitive.
                   18.   Radio button primitive.
            19.   Checkbox primitive.
            20.   Static text primitive.
21.   Group box control (kControlGroupBoxSecondaryTextTitleProc variant).
       22.   Static text primitive.
23.   Group box control (kControlGroupBoxSecondaryTextTitleProc variant).
       24.   Static text primitive.
       25.   Placard control.
              26.   Static text primitive.
       27.   Little arrows control.
28.   Extract push button primitive.
29.   Image well
30.   Separator line
31.   Disclosure triangle (kControlTriangleProc variant).
32.   Static text primitive.
```

A second dialog item list resource, containing a chasing arrows control and a progress bar control, is appended to the dialog when the disclosure triangle is clicked.

doGroupArrowsProgress

doGroupArrowsProgress creates the dialog, calls ModalDialog to handle events until the user hits the OK push button, and then disposes of the dialog.

At the first two lines, if the Bevel Buttons, Image Wells, Picture Controls, and Icon Controls window is open, it is explicitly deactivated.

The call to GetNewDialog creates the dialog and SetDialogDefaultItem establishes the OK push button as the default push button.

The next two lines create universal procedure pointers for the application-defined event filter (callback) function and an application-defined action function for the little arrows control. (The event filter function is in the source code file Callbacks.c.)

The next block sets the initial value of the checkbox group box to 1 (checked).

The little arrows are used to set a (simulated) cache size, which is displayed in a static text item overlaid on a placard item. The little arrows initial value is set in the `'CNTL'` resource to 96. The first three lines of the next block extract this value, convert it to a string and append the character `"K"`. `SetControlData` is then called with the `kControlStaticTextTextTag` to set this string in the static text item.

The second user pane in the pop-up group box is then hidden before the call to `ShowWindow`, following which the `ModalDialog` loop is entered. Note that the UPP associated with the application-defined event filter function is passed in the `filterProc` parameter of `ModalDialog`.

If `ModalDialog` reports that the item hit was the checkbox group box's checkbox or the pop-up group box's pop-up menu, functions are called to further handle the hit.

If the item hit was one of the checkboxes within the pop-up group box, the control's control value is set so that the control is checked if it was unchecked, or unchecked if it was checked.

If the item hit was the disclosure triangle, a function is called to further handle the hit.

If the item hit was the Extract push button, `Draw1Control` is called to redraw the image well, thus erasing any previous text in the well, and a function is called to extract the control settings and draw them in the image well area.

Note that hits on the radio buttons will be handled automatically by the radio group controls in which they are embedded. The radio group control will set the new control values according to which radio button is hit, and will change the checked/unchecked appearance of the radio buttons to match.

Note also that mouse-down events in the little arrows are detected and handled in the event filter function. (See the source code file `Callbacks.c`.)

When the OK button is hit, the loop exits, `DisposeDialog` closes the dialog and disposes of the associated memory, and the two previously created universal procedure pointers are disposed of. In addition, the global variable that flags that the dialog was open is set to false.

doCheckBoxGroupBox

`doCheckBoxGroupBox` further handles a hit in the checkbox group box's checkbox.

The first three lines flip the control's value; that is, if the control's value was 1 (checked), it is set to 0 (unchecked), and vice versa.

If the new value of the control is 0 (unchecked), the radio group and checkbox within the group box are deactivated. Since the radio buttons are embedded in the radio group, they will also be deactivated when the radio group is deactivated.

If the new value of the control is 1 (checked), the radio group and checkbox within the group box are activated. Since the radio buttons are embedded in the radio group, they will also be activated when the radio group is activated.

doPopupGroupBox

`doPopupGroupBox` further handles a hit in the pop-up group box's pop-up menu button.

The first two lines get the control's value, which represents the menu item chosen. Depending on the item chosen, the appropriate user panes are hidden or shown, thus hiding or showing all controls embedded in each user pane.

doChasingAndProgress

`doChasingAndProgress` further handles a hit in the disclosure triangle.

The first three lines flip the control's value; that is, if the control's value was 1 (expanded), it is set to 0 (collapsed), and vice versa.

If the new control value is 1 (expanded), `GetResource` is called to load the item list resource containing the chasing arrows and progress bar items. `AppendDITL` is then called to append these items to the end of the dialog's item list. The constant passed in the

third parameter of this call causes the dialog to be expanded downwards to accommodate the new items. `SetControlData` is then called with the `kControlProgressBarIndeterminateTag` tag to make the progress bar an indeterminate progress bar.

Not previously mentioned is the fact that the appended 'DITL' resource contains a static text item, with no text, the function of which is simply to expand the dialog further downwards to accommodate a move of the OK push button. The call to MoveControl moves the OK push button to a position below the chasing arrows and progress bar.

Finally, `SetControlData` is called with the `kControlStaticTextTextTag` tag to change the text in the static text control adjacent to the disclosure triangle. `Draw1Control` redraws the control to display the changed text.

If the new disclosure triangle control value is 0 (collapsed), the OK button is moved back to its previous location, `SizeWindow` is called to resize the window to its previous size, `SetControlData` is called with the `kControlStaticTextTextTag` tag to change the text in the static text control adjacent to the disclosure triangle, and `Draw1Control` redraws the control to display the changed text.

doExtractCurrentStatus

`doExtractCurrentStatus` is called when the Extract push button is hit. It simply extracts the values of the various controls, builds up strings based on those values, and draws those strings in the image well area.

Note in the second last block that the value of the little arrows control represents the current (simulated) cache size. The little arrows control value is set within the action function `arrowsActionFunction` in the source code file `Callbacks.c`.

Sliders.c

`Sliders.c` creates a movable modal dialog in which is demonstrated various slider control variants and two simple user pane functions.

doSliderUserPane

`doSliderUserPane` creates the dialog, calls `ModalDialog` to handle events until the user hits the OK push button, and then disposes of the dialog.

At the first two lines, if the Bevel Buttons, Image Wells, Picture Controls, and Icon Controls window is open, it is explicitly deactivated.

The call to `GetNewDialog` creates the dialog and `SetDialogDefaultItem` establishes the OK push button as the default push button.

The next three lines create universal procedure pointers for the application-defined event filter (callback) function and two application-defined action functions for the live-feedback slider variants, one for the two at top right of the dialog and one for the two horizontal sliders. (The event filter function and the action functions are in the source code file `Callbacks.c`.)

The next two blocks creates universal procedure pointers for two user pane functions (one a draw function and one an activate function), and attach these functions to a user pane located immediately to the right of the two horizontal sliders.

The next block assigns references to the first four sliders to global variables and call a function to draw the respective control values in static text items located immediately below the sliders.

The following four lines assign references to the horizontal sliders to global variables and assign a value based on the current slider control values to the red and blue fields of two global variables of type RGBColor.

The call to `ShowWindow` makes the dialog visible before the `ModalDialog` loop is entered. This loop executes until the OK button is hit, at which time `DisposeDialog` is called to remove the dialog and dispose of its associated memory, the previously created universal procedure pointers are disposed of, and the global variable which flags that the Sliders dialog is open is assigned false.

doDrawSliderValues

`doDrawSliderValue` is called by `doSliderUserPane` to draw the initial control values of the four top sliders in the static text controls immediately below the sliders.

The first line gets the control's value and converts it to a string. The next block determines, on the basis of the received control reference, which is the target static text item. `SetControlData` is then called with the `kControlStaticTextTextTag` tag to set the text, and `Draw1Control` redraws the static text control.

As will be seen, this function is also called from within the event filter function when a mouse-down has been detected within the two non-live-scrolling sliders.

userPaneDrawFunction

`userPaneDrawFunction` will be called whenever an update event is received. This will occur when the dialog is opened and, subsequently, when an overlaying window or help balloon is removed from the area occupied by the user pane (to the immediate right of the horizontal sliders).

The two calls to `DrawThemePlacard` draw two Appearance Manager placard primitives to the right of the horizontal sliders. The global variable `gDrawActivated` determines whether the primitive is drawn with an activated or deactivated appearance.

If the global variable `gDrawActivated` indicates that the interior of the placard should be drawn, the foreground color for each interior is set to that stored in global variables and `PaintRect` is called to paint the interior in that color. (As will be seen, the value stored in these global variables is changed by the action function called while the mouse button remains down in the slider's indicators.)

If `gDrawActivated` indicates that the interior of the placard should be displayed with a deactivated appearance, the interior is left as drawn by `DrawThemePlacard`.

userPaneActivateFunction

The `kControlWantsActivate` feature bit is set in the user pane on creation, that is, the control's minimum value in the `'CNTL'` resource is set to 16. This ensures that this function will be called when the dialog receives activate events.

Depending on whether the dialog is about to be activated or deactivated, the global variable `gDrawActivated` is set to true or false. As has been seen, this global variable controls the way in which the placard primitive and its interior is drawn.

TextBox.c

`doTextBox` is called when the user chooses `TextBoxes` from the Demonstration menu. It opens a movable modal dialog containing standard and scrolling text box controls.

Callbacks.c

`Callbacks.c` contains the event filter (callback) function used by all movable modal dialogs and three action functions, one for the little arrows, one for the vertical live-feedback sliders, and one for the horizontal live-feedback sliders.

eventFilter

The event filter function `eventFilter` is identical to the event filter function introduced in Chapter 8 except that it intercepts mouse-down events in the little arrows and sliders and informs `ModalDialog` that it has handled those events.

After the call to `SetPortDialogPort`, if the Group Boxes, Arrows, and Progress Bar dialog is currently open, and if the event is a mouse-down event, the location of the mouse-down is extracted from the where field of the event structure and converted to the local coordinates required by the following call to `FindControl`. If `FindControl` determines that there is an enabled control under the mouse cursor, and if that control is the little arrows control, `TrackControl` is called to handle all user interaction while the mouse button remains down. While the mouse button remains down, `TrackControl` continually calls the action function pointed to by the UPP passed in the third parameter. When the mouse button is released, the function exits with true being returned to `ModalDialog`.

If, on the other hand, the Sliders and User Pane Functions dialog is open, and if the event is a mouse-down event, and if `FindControl` reports an enabled control, and if that control is one of the two non-live-feedback sliders, `TrackControl` is called to handle all user interaction while the mouse button remains down. When the mouse button is released, the function `doDrawSliderValues` is called to draw the new control value in the static text item below the slider. Once again, the function then exits with true being returned to `ModalDialog`.

If the mouse-down was in one of the two live-feedback sliders at the top right of the dialog, `TrackControl` is called with a UPP to

the first slider action function passed in the third parameter.

If the mouse-down was in one of the two horizontal live-feedback sliders, `TrackControl` is called with a UPP to the second slider action function passed in the third parameter.

arrowsActionFunction

`arrowsActionFunction` is the action function for the little arrows. It is called continually by `TrackControl` while the mouse button remains down.

If the mouse cursor is still within the control, `GetControlValue` is called to get the current control value. The function then switches according to the part code. If the cursor is in the top arrow, the control's value is increased by 32 unless this would cause the value to exceed the control's maximum value (set in the `'CNTL'` resource to 7680). If the cursor is in the bottom arrow, the control's value is decreased by 32 unless this would cause the value to be lower that the control's minimum value (set in the `'CNTL'` resource to 96).

If the control's value was increased or decreased within the switch, `SetControlValue` is called to set the new control value, and the final block sets the new text for the Cache size static text item and re-draws the item.

sliderActionFunction1

`sliderActionFunction1` is the action function for the live-feedback sliders at the top right of the dialog. It is called continually by `TrackControl` while the mouse button remains down.

`GetControlValue` gets the control's current value and `NumToString` converts it to a string for display. The next line gets the reference to the control's owning window required by the call to `GetDialogItemAsControl`. The if block determines which of the two sliders the mouse is down in. The last two lines set the text in the appropriate static text control and redraw the control.

sliderActionFunction2

`sliderActionFunction3` is the action function for the horizontal live-feedback sliders. It is called continually by `TrackControl` while the mouse button remains down.

`GetControlValue` gets the control's current value. If the slider is the top horizontal slider, the red field of the `RGBColor` global variable `gRedColor` is assigned a new value based on the control's current value, and the local `Rect` variable is assigned coordinates based on the placard at the right of the slider. If the slider is the bottom horizontal slider, the blue field of the `RGBColor` global variable `gBlueColor` is assigned a new value based on the control's current value, and the local `Rect` variable is assigned coordinates based on the placard at the right of the slider.

Finally, `PaintRect` is called to paint the interior area of the relevant image well in the new color.

SmallControls.c

doSmallControls

`doSmallControls` is called when the user chooses the Small Controls item in the Demonstration menu (Mac OS X only). It opens a floating window in which those controls with small versions are displayed.

Note that

- A call to `SetControlData` with the `kControlSizeTag` tag is required to make the scroll bars small.
- The value passed in the third parameter of the `CreateTabsControl` call makes the tab control small.
- A call to `SetControlData` with the `kControlSizeTag` tag is required to make the slider control small.
- A call to `SetControlData` with the `kControlSizeTag` tag is required to make the radio button and checkbox controls small. In addition, a call to `SetControlFontStyle` is required to make the title small.
- The only way to create a pop-up menu button with both the menu and the title small is to create the control from a `'CNTL'` resource, specifying the "use window font" variant. (Note that the window font is set to the small system font.)
- The edit text control is made small by making the control's rectangle 12 pixels high and changing the font to the small system font by passing the address of a `ControlFontStyleRec` structure in the sixth parameter of the `CreateEditTextControl` call.
- The push button is made small by making the control's rectangle 17 pixels high and calling `SetControlFontStyle` is to

make the title small.

Chapter 15. Carbon Printing

IN THIS CHAPTER

- The Carbon Printing Manager
- Printer Drivers
- Page and Paper Rectangles
- Page Setup Dialogs and Print Dialogs
- Printing Sessions-The PMPrintSession Object
- The PMPageFormat and PMPrintSettings Objects
- Printing a Document
- Text on the Screen and the Printed Page
- Customizing the Page Setup and Print Dialogs
- Displaying Page Setup and Print Dialogs As Window-Modal (Sheet) Dialogs
- Saving and Retrieving a Page Format Object
- Printing from the Finder-Mac OS 8/9
- Main Carbon Printing Manager Constants, Data Types and Functions
- Demonstration Program CarbonPrinting Listing
- Demonstration Program CarbonPrinting Comments

The Carbon Printing Manager

The inclusion of the word "Carbon" in the title of this chapter is quite deliberate, reflecting the fact that, in Carbon, the original **Printing Manager** has been replaced by the new **Carbon Printing Manager.** The Carbon Printing Manager provides a bridge between the fundamentally different Mac OS X and Mac OS 8/9 printing architectures.

When running on Mac OS 8/9, Carbon applications use Classic printer drivers and, generally speaking, Carbon Printing Manager functions simply call through to their original Printing Manager counterparts. When running on Mac OS X, Carbon applications use the new printing architecture and print through different drivers.

The most significant difference between the old Printing Manager and the Carbon Printing Manager is that all data structures are now opaque. Accessor functions are provided to access the information in these objects.

Printing Sessions

In Carbon Printing Manager parlance, an individual printing task is known as a **session.** Carbon applications running on Mac OS X can initiate multiple simultaneous printing sessions, each of which is completely independent of the other. On Mac OS 8/9, printing sessions are also supported, but with the limitation that an application can only ever have one printing session running.

Categories of Carbon Printing Manager Functions

The Carbon Printing Manager API comprises

- Session functions
- Non-session functions
- Universal functions

The non-session functions do not support simultaneous printing sessions and inherit many of the limitations of the original Printing Manager. For this reason, Apple strongly recommends that the session functions be used instead. (Session and non-session functions must not be mixed within an application.) Accordingly, this chapter addresses the session and universal functions only.

Printer Drivers

Each type of printer has its own **printer driver.** Printer drivers performs the actual printing, translating system software drawing functions as required and send the translated instructions and data to the printer. On Mac OS 8/9, printer drivers are stored in **printer resource files.**

The **current printer** (on Mac OS 8/9, the printer selected by the user in the Chooser) is the printer driver that actually implements the functions defined by the Carbon Printing Manager.

Types and Characteristics of Printer Drivers

In general, there are two types of printer driver:

- QuickDraw printer drivers
- PostScript printer drivers

QuickDraw Printers

QuickDraw printers use QuickDraw to render images, which are then sent to the printer as bitmaps or pixel maps. Since they rely on the rendering capabilities of the Macintosh computer, QuickDraw printers are not required to have any intelligent rendering capabilities. Instead, they simply accept instructions from the printer driver to place dots on the page in specified places.

PostScript Printers

Unlike QuickDraw printers, PostScript printers have their own rendering capabilities. Instead of rendering the entire page on the Macintosh computer and sending all the pixels to the printer, PostScript printer drivers convert QuickDraw operations into equivalent PostScript operations and send the resulting drawing commands directly to the printer. The printer then renders the images by interpreting these commands. In this way, image processing is offloaded from the computer.

Background Printing and Spool Files

Most printer drivers allow users to specify **background printing,** which allows a user to work with an application while documents are printing in the background. On Mac OS 8/9, these printer drivers send printing data to a spool file in the PrintMonitor Documents folder

in the System Folder.

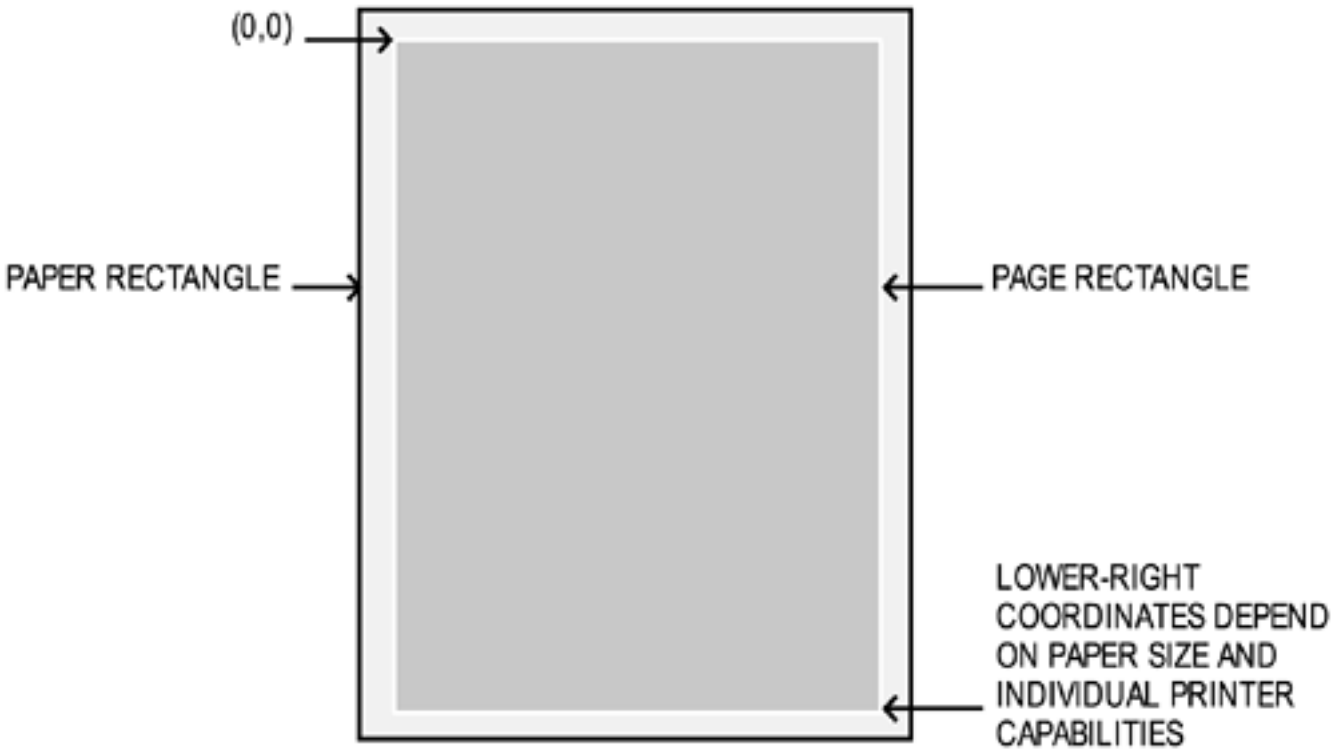
Page and Paper Rectangles

Because of an individual printer's mechanical limitations, the printable area of a page is ordinarily less than the physical size of the paper.

Page Rectangle

The **page rectangle** represents the printable area. As shown in Fig 1, the upper-left coordinates of the page rectangle are always (0,0) and the lower-right coordinates represent the maximum printable area height and width for a given printer.

Figure 1. PAGE AND PAPER RECTANGLES



Paper Rectangle

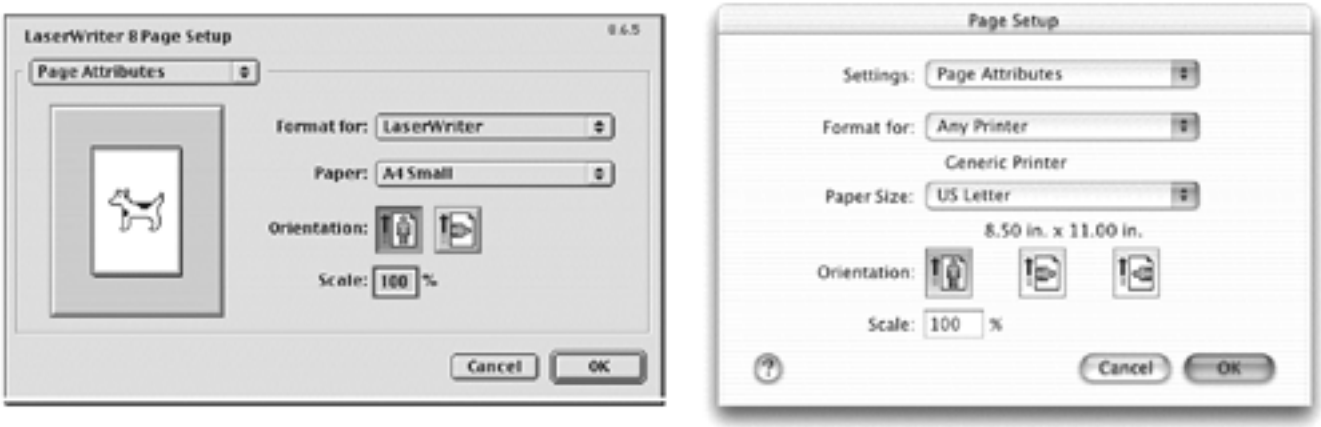
The **paper rectangle** (see Fig 1) represents the physical paper size expressed in the same coordinate system as the page rectangle. The upper left coordinates of the paper rectangle are thus usually negative.

Page Setup Dialogs and Print Dialogs

Page Setup Dialog

In response to the user choosing the Page Setup… item from the File menu, your application should display the **Page Setup dialog.** For Mac OS 8/9, each printer driver defines its own Page Setup dialog. For Mac OS X, printer manufacturers can extend the standard Page Setup dialog with options specific to their printer. Fig 2 shows the Page Setup dialog.

Figure 2. PAGE SETUPDIALOG



LASERWRITER 8 PRINTER — MAC OS 8/9 MAC OS X

Displaying the Page Setup Dialog and Accessing Settings

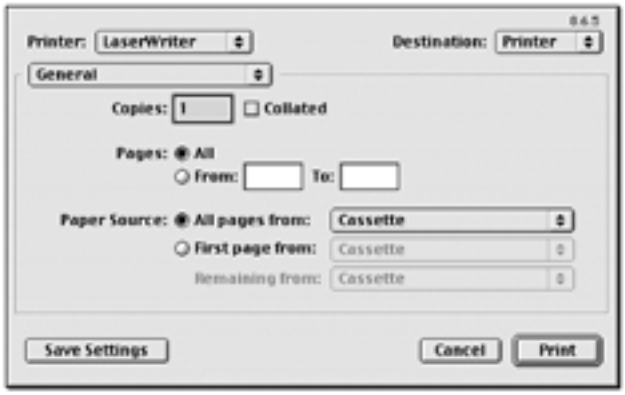
`PMSessionPageSetupDialog` is used to display the Page Setup dialog. This function handles all user interaction until the user clicks the OK or Cancel button.

Settings made in a Page Setup dialog are stored in a `PMPageFormat` object. Your application can use accessor functions to extract information, such as orientation and scaling settings, from this object.

Print Dialog

In response to the user choosing the Print… item in the File menu, your application should display the **Print dialog.** For Mac OS 8/9, each printer driver defines its own Print dialog. For Mac OS X, printer manufacturers can extend the standard Print dialog with options specific to their printer. Fig 3 shows the Print dialog.

Figure 3. PRINT DIALOG



LASERWRITER 8 PRINTER — MAC OS 8/9



MAC OS X

Displaying the Print Dialog and Accessing Settings

`PMSessionPrintDialog` is used to display the Print dialog. This function handles all user interaction until the user clicks the Print or Cancel button.

Settings made in a Print dialog are stored in a `PMPrintSettings` object. Your application can use accessor functions to extract information, such as the page numbers to print, from this object.

Customized Page Setup and Print Dialogs

Many applications add items to the basic Page Setup and Print dialogs so as to provide the user with additional control over printing operations within that application.

If you wish to customize the Page Setup and/or Print dialogs so as to solicit additional information from the user, one option is to use what is sometimes referred to as the AppendDITL method. This requires that you provide an **initialization function,** an **item evaluation function** and, possibly, an event filter function. A universal procedure pointer to the initialization function is passed as a parameter in the functions `PMSessionPageSetupDialogMain` and `PMSessionPrintDialogMain`, which are used to display, respectively, customized Page Setup and customized Print dialogs. (See "Customizing the Page Setup and Print Dialogs.")

Preserving the User's Printing Settings

The only information you should preserve each time the user prints the document should be that obtained via the Page Setup dialog. The information supplied by the user through the Print dialog should pertain to the document only while the document prints, and you should not reuse this information if the user prints the document again.

You can preserve the information obtained via the Page Setup dialog by associating the `PMPageFormat` object with the relevant document's window and by saving it to that

document file's data or resource fork when the file is closed. (See "Saving and Retrieving a Page Format Object.")

Printing Sessions-The PMPrintSession Object

A call to `PMCreateSession` creates a context for printing operations, called a printing session, and initializes a `PMPrintSession` object.

The PMPageFormat and PMPrintSettings Objects

PMPageFormat Object

The `PMPageFormat` object stores information about how the pages of a document should be printed, for example, on what paper size and in what orientation (landscape or portrait).

You use the function `PMCreatePageFormat` to create an instance of this opaque object. `PMCreatePageFormat` returns a reference to the object, which is created empty of settings. When your application displays a Page Setup dialog, and the user clicks the OK button to dismiss the dialog, the Carbon Printing Manager saves the settings in the `PMPageFormat` object.

As will be seen, `PMPageFormat` objects are extensible, meaning that your application can store additional data in them and that your application should never assume that they are of a fixed size.

Accessor Functions

The following describes accessor functions you can use to obtain information contained within `PMPageFormat` objects.

Function	*Description*

`PMGetAdjustedPaperRect`	Gets the paper size.
	
	On return, the `paperRect` parameter contains a rectangle describing the size of the paper after scaling, application drawing resolution and orientation settings are applied. The `paperRect` parameter is of type `PMRect`:
	
	`struct PMRect`
	`{`
	`  double top;`
	`  double left;`
	`  double right`
	`  double bottom;`
	`}`
	
	The size returned is in your application drawing resolution, which you can obtain by calling `PMGetResolution`.

`PMGetAdjustedPageRect`	Gets the page size. On return, the `pageRect` parameter contains a rectangle describing the size of the page after scaling, application drawing resolution, and orientation settings are applied. The `pageRect` parameter is of type `PMRect`. The size returned is in your application drawing resolution, which you can obtain by calling `PMGetResolution`.
`PMGetUnadjustedPaperRect`	Gets the true size of the paper in points, unaffected by rotation, resolution, or scaling.

`PMSetUnadjustedPaperRect`	Sets printing to a particular paper size, in points, unaffected by rotation, resolution, or scaling. This function allows applications to request a particular paper size. If the driver cannot handle the specified size an error of `kPMValueOutOfRange` is returned. If the size is accepted, the application should still call `PMGetUnadjustedPaperRect` to verify.
`PMGetUnadjustedPageRect`	Gets the size of the imageable area in points, unaffected by rotation, resolution, or scaling.

`PMGetResolution`	Gets the application's current drawing resolution.
	
	On return, the `res` parameter contains a pointer to a structure of type `PMResolution`, which describes the resolution at which the Carbon Printing Manager expects your application to render images:
	
	`struct PMResolution`
	`{`
	`  double hRes;  // The horizontal resolution in dpi.`
	`  double vRes;  // The vertical resolution in dpi.`
	
	`} ;`

`PMSetResolution`	Sets the application's drawing resolution.

`PMGetOrientation`	Gets the current page orientation setting.
	
	On return, the `orientation` parameter contains a pointer to a variable of type `PMOrientation`, which describes the current page orientation:
	
	`kPMPortrait         = 1`
	`kPMLandscape        = 2`
	`kPMReversePortrait  = 3`
	`kPMReverseLandscape = 4`

`PMSetOrientation`	Sets the page orientation for printing.
`PMGetScale`	Returns the scaling factor currently applied to the page and paper rectangles.
`PMSetScale`	Sets print scaling.
`PMGetPageFormatExtendedData`	Gets additional page format data previously stored by your application.
`PMSetPageFormatExtendedData`	Stores application-supplied data in a `PMPageFormat` object.

Assigning Default Parameters

A call to `PMSessionDefaultPageFormat` will assign default parameters to a `PMPageFormat` object for the specified printing session.

Validating a PMPageFormat Object

A call to `PMSessionValidatePageFormat` will validate a `PMPageFormat` object's parameters within the context of the specified printing session. `true` is returned in the `result` parameter if any parameters had to be changed.

PMPrintSettings Object

The `PMPrintSettings` object stores information such as the number of copies, range of pages to print, and so on, for a particular printing session.

You use the function `PMCreatePrintSettings` to create an instance of this opaque object. `PMCreatePrintSettings` returns a reference to the object, which is created empty of settings. When your application displays a Print dialog, and the user clicks the Print button to dismiss the dialog, the Carbon Printing Manager saves the settings in the `PMPrintSettings` object.

As will be seen, `PMPrintSettings` objects are extensible, meaning that your applications can store additional data in them and that your application should never assume that they are of a fixed size.

Accessor Functions

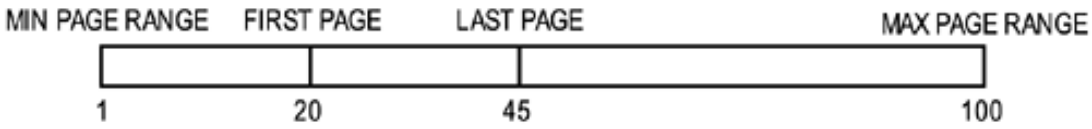
The following describes the accessor functions you can use to obtain information contained within `PMPrintSettings` objects.

Function	*Description*
`PMGetFirstPage`	Gets the starting page number of the pages to be printed.
`PMSetFirstPage`	Sets the page number of the first page to be printed.
`PMGetLastPage`	Gets the last page number of the pages to be printed.
`PMSetLastPage`	Sets the page number of the first page to be printed.
`PMGetPageRange`	Gets the valid range of pages to print, as previously set by `PMSetPageRange`. If none was set by the application, the default range (1-32000) is returned.
`PMSetPageRange`	Sets the valid range of pages to be printed. On Mac OS X, if the user enters a value outside this range in the Print dialog, an alert message is displayed. This feature is not supported on Mac OS 8/9. The relationship between the page range set by the application using `PMSetPageRange` and the first and last pages set by the user in the Print dialog (and retrieved by `PMGetFirstPage` and `PMGetLastPage`) is shown in Fig 4.
`PMGetCopies`	Gets the number of copies to be printed.
`PMSetCopies`	Sets the default value to be displayed for the number of copies to be printed.
`PMGetPrintSettingsExtendedData`	Gets additional print settings previously saved by your application.

`PMSetPrintSettingsExtendedData`	Stores application-supplied data in a `PMPrintSettings` object.

Note: You pass `true` in the `lock` parameter of the functions `PMSetCopies`, `PMSetFirstPage`, and `PMSetLastPage` if you wish to lock, respectively, the number-of-copies, first-page, and last-page fields in the Print dialog. Passing `true` only affects printer drivers for Mac OS X and LaserWriter printer drivers version 8.7 and later. If you pass `true` for other printer drivers, these functions will return `kPMLockIgnored`.

Figure 4. FIRST PAGE, LAST PAGE, AND PAGE RANGE (EXAMPLE)



Assigning Default Parameters

A call to `PMSessionDefaultPrintSettings` will assign default parameters to a `PMPrintSettings` object for the specified printing session.

Validating a PMPrintSettings Object

A call to `PMSessionValidatePrintSettings` will validate a `PMPrintSettings` object's parameters within the context of the specified printing session. `true` is returned in the `result` parameter if any parameters had to be changed.

Printing a Document

The following describes a typical approach to printing a document.

When the User Chooses Page Setup… from the File Menu

When the user invokes the Page Setup dialog

- Call a function which
    - Calls `PMCreateSession` to create a `PMPrintSession` object, and associates that object with the document's window.
    - If a `PMPageFormat` object has not previously been created and associated with the document's window
        - Calls PMCreatePageFormat to create a PMPageFormat object
        - Calls `PMSessionDefaultPageFormat` to assign default parameters to the `PMPageFormat` object
        - Associates the `PMPageFormat` object with the document's window
    - If a `PMPageFormat` object has previously been created and associated with the document's window, calls `PMSessionValidatePageFormat` to validate the `PMPageFormat` object's parameters within the context of this printing session.
- Call `PMSessionPageSetupDialog` to present the Page Setup dialog, handle all user interaction within the dialog, and record the user's settings in the `PMPageFormat` object.
- When the user dismisses the Page Setup dialog, call `PMRelease` to release the `PMPrintSession` object.

When the User Chooses Print… from the File Menu

When the user invokes the Print dialog

- As when the Page Setup dialog is invoked, call a function which
    - Creates a `PMPrintSession` object and associates it with the document's window.
    - Either creates a `PMPageFormat` object, assigns default parameters to it, and associates it with the document's window or, if this has previously been done, validates the existing `PMPageFormat` object's parameters.
- Call `PMCreatePrintSettings` to create a `PMPrintSettings` object.
- Call `PMSessionDefaultPrintSettings` to assign default parameters to the `PMPrintSettings` object.
- Associate the `PMPrintSettings` object with the document's window.
- For Mac OS X only
    - Call `PMSetPageRange` to set the valid range of pages that can be printed.
    - Call `PMSetFirstPage` and `PMSetLastPage` to set the default first and last pages to be printed, as displayed in the From and To fields of the Print dialog. (To clear the From and To fields and select the All radio button, pass `kPMPrintAllPages` in the `PMSetLastPage` call.)
- Call `PMSessionPrintDialog` to present the Print dialog, handle all user interaction within the dialog, and record

the user's settings in the `PMPrintSettings` object.

- If the user clicks the Cancel button in the Print dialog, call `PMRelease` to release the `PMPrintSettings` and `PMPrintSession` objects, and disassociate them from the document's window.
- If the user clicks the Print button in the Print dialog, call your application's **printing loop** function.

The Printing Loop Function

The printing loop is the part of your application's code that performs the actual printing. The function containing the printing loop should perform the following actions:

- Call `PMGetFirstPage` and `PMGetLastPage` to get the first and last pages to print, as entered by the user in the Print dialog.
- Call a function which calculates the actual number of pages in the document. If necessary, adjust the value in the variable containing the last page as returned by `PMGetLastPage`.
- For Mac OS X, call `PMSetFirstPage` and `PMSetLastPage` to tell the Carbon Printing Manager which pages will be spooled so that the progress dialog can display an accurate page count. Pass in the values returned by the `PMGetFirstPage` and `PMGetLastPage` calls, the latter adjusted as necessary.
- Call `PMSessionBeginDocument` to create an instance of a `PMPrintContext` object and thus establish a graphics context for imaging the document. (As will be seen, a function exists to obtain the graphics port (that is, the printing port) associated with this object.)
- Print the pages. In the page-printing loop, all of the pages in the document should be spooled and the Carbon Printing Manager relied upon to print the correct page range as specified in the Print dialog. Note that your printing loop does not have to concern itself with the number of copies, since this is handled automatically by the Carbon Printing Manager. The pages loop should
    - Call `PMSessionBeginPage` to initialize the printing port. (Note that, for Mac OS 8/9 only, you can pass a scaling (bounding) rectangle in the `pageFrame` parameter, in which case all items drawn in the printing port will be scaled to fit this page rectangle. If no scaling is required, pass `NULL` in the `pageFrame` parameter.)
    - Call `GetPort` to save the current graphics port, call `PMSessionGetGraphicsContext` to obtain the current printing port, and call `SetPort` to set that port as the current port.
    - Call a function which draws the relevant page in the printing port, then call `SetPort` to set the port saved by the `GetPort` call as the current port.
    - Call `PMSessionEndPage` to print the page.
- When all copies of all pages have been printed, call `PMSessionEndDocument` to close the printing graphics port.
- Call `PMRelease` to release the `PMPrintSettings` and `PMPrintSession` objects.

Call Sequence and Scope

When writing functions that call Carbon Printing Manager functions, bear in mind that the Carbon Printing Manager enforces a **sequence** of steps in the printing loop, and defines a valid **scope** for each Carbon Printing manager function. Functions used out of sequence will return a result code of `kPMOutOfScope`.

Sequence and Scope: Session Functions

The following, in which scope level is represented by indentation, shows calling sequence and scope requirements of the main session printing functions. It shows, for example, that you can call `PMSessionGetGraphicsContext` only after calling

```
PMSessionBeginPage.

PMCreateSession
   PMSessionDefaultPageFormat
   PMSessionValidatePageFormat
   PMSessionDefaultPrintSettings
   PMSessionValidatePrintSettings
   PMSessionError

   PMSessionPageSetupDialog
   PMSessionPrintDialog
   PMSessionPageSetupDialogInit              These functions must be called
   PMSessionPrintDialogInit                  before PMSessionBeginDocument
   PMSessionPrintDialogMain
   PMSessionPageSetupDialogMain

   PMSessionBeginDocument
      PMSessionBeginPage
         PMSessionGetGraphicsContext
      PMSessionEndPage
   PMSessionEndDocument
```

In general, functions may be called in any order with respect to other functions at the same or lower scope level.

Sequence and Scope: Universal Functions

The following are the main universal functions which have no calling sequence or scope, and which may generally be used anywhere in your printing code:

```
PMCreatePageFormat          PMGetLastPage              PMGetScale
PMFlattenPageFormat         PMSetLastPage              PMSetScale
PMUnflattenPageFormat       PMGetCopies                PMGetResolution
PMCreatePrintSettings       PMSetCopies                PMSetResolution
PMFlattenPrintSettings      PMGetAdjustedPaperRect     PMGetPageFormatExtendedData
PMUnflattenPrintSettings    PMGetAdjustedPageRect      PMSetPageFormatExtendedData
PMGetPageRange              PMGetUnadjustedPageRect    PMGetPrintSettingsExtendedData
PMSetPageRange              PMSetUnadjustedPaperRect   PMSetPrintSettingsExtendedData
PMGetFirstPage              PMGetOrientation
PMSetFirstPage              PMSetOrientation
```

Handling Printing Errors

The Carbon Printing Manager must necessarily bear the heavy burden of maintaining backward compatibility with early printer models and of maintaining compatibility with a great many existing printer drivers. For this reason, you must be especially wary of, and defensive about, possible error conditions when using Carbon Printing Manager functions.

Do not display an alert to report an error until the end of the printing loop. This is important for two reasons:

- If you display an alert in the middle of a printing loop, it could cause errors that might terminate an otherwise normal printing operation.
- The printer driver may have already displayed its own alert reporting the error.

Text on the Screen and the Printed Page

At the application level, printing on the Macintosh computer is not fundamentally different from drawing on the screen. That said, printing text poses special challenges.

A common complication results from the difference in resolution and pixel size between screen and printer. QuickDraw measurements are theoretically in terms of **points,** which are nominally equivalent to screen pixels. High resolution printers have very much smaller pixels, although printer drivers are expected to take this into account so that the same QuickDraw calls will produce text lines of the same width on the screen and on the printer. Nevertheless, this higher resolution, and the fact that printers may use different fonts from those used for screen display, can result in some loss of fidelity from the screen to the printed page. In this regard, the following is relevant:

- QuickDraw places text glyphs on the screen at screen pixel intervals, whereas a printer can provide much finer placements on the printed page. This situation presents a choice between optimizing the appearance of text on the screen or on the printed page. In effect, that choice is whether to specify **fractional glyph widths** or **integer glyph widths.**

  Note

  A glyph is the visual representation of a character. See Chapter 21.

  Fractional glyph widths are measurements of a glyph's width which can include fractions of a pixel. Using fractional glyph widths improves the appearance of printed text because it makes it possible for the printer, with its very high resolution, to print with better spacing. However, because screen glyphs are made up of whole pixels, QuickDraw cannot draw a fractional glyph on the screen, so it rounds off the fractional parts. This results in some degradation in the appearance of the text, in terms of character spacing, on the screen.

  The alternative (integer glyph widths) gives more pleasing screen results because the characters are drawn with regular pixel spacing, but this may possibly be at the price of a printed page which is typographically unacceptable.

  The Font Manager function `SetFractEnable` is used to turn fractional glyph widths on and off. `SetFractEnable` affects functions which draw text and which calculate text and character widths.

- Printer drivers attempt to reproduce faithfully the text formatting as drawn by QuickDraw on the screen, including keeping the same intended character spacing, line breaks and page breaks. However, because printers can have resident fonts that are different from the fonts that QuickDraw uses, because the drivers may handle text layout somewhat differently than QuickDraw, and because font metrics do not always scale linearly, fidelity may not always be achieved. Typically, identical line breaks and page breaks can be maintained, but character spacing can be noticeably different.

Customizing the Page Setup and Print Dialogs

As previously stated, you may want to add additional options to the Page Setup and Print dialogs so that the user can further customize the printing process. For example, you might want to add a "skip blank pages" checkbox to a Print dialog.

> **Note**
>
> On Mac OS X, the **printing dialog extension** (PDE) mechanism may be used to extend the Page Setup and Print dialogs. The PDE mechanism provides great flexibility in extending printing dialogs. However, because the PDE mechanism applies only to Mac OS X, that method of extending Page Setup and Print dialogs is not addressed in this book. The method addressed is sometimes referred to as the AppendDITL method.
>
> A limitation of the AppendDITL method is that it prevents you from creating the Page Setup and Print dialogs as window-modal (sheet) dialogs. This limitation does not apply when the PDE method is used.

The functions `PMSessionPageSetupDialogMain` and `PMSessionPrintDialogMain` are used to display Page Setup and Print dialogs customized using the AppendDITL method.

The PMDialog Object

Your application uses a reference to a `PMDialog` object when creating custom Page Setup and Print dialogs. The functions `PMSessionPageSetupDialogInit` and `PMSessionPrintDialogInit` functions are used to create and initialize instances of this opaque object. Taking the reference to this object as a parameter, the function `PMGetDialogPtr` returns a pointer to the dialog structure.

Customizing a Print Dialog

As an example, to customize a Print dialog, you must modify the contents of the `PMDialog` object before the dialog is drawn on the screen. This involves

- Providing a `'DITL'` resource containing the required additional items.

- Defining an **item evaluation function** that handles events involving the additional items.

- Defining and installing an **initialization function** that

    - Calls `AppendDITL` to append the additional items to the dialog.

    - Using `PMGetItemProc`, gets the universal procedure pointer to the printer driver's item evaluation function from the `PMDialog` object, and saves it. (The printer driver's item evaluation function will need to be called from your item evaluation function to handle hits on the dialog's standard items.)

    - Calls `PMSetItemProc` to set a universal procedure pointer to your item evaluation function in the `PMDialog` object.

- If required, defining a custom event filter function and setting a universal procedure pointer to it in the `PMDialog` object using `PMSetModalFilterProc`.

A universal procedure pointer to the initialization function should then be passed in the `myInitProc` parameter of `PMSessionPrintDialogMain`, which displays the customized Print dialog.

Displaying Page Setup and Print Dialogs As Window-Modal (Sheet) Dialogs

To cause Page Setup and Print dialogs to be created as window-modal (sheet) dialogs on Mac OS X, you should call the function `PMSessionUseSheets` immediately before the calls to `PMSessionPageSetupDialog` and `PMSessionPrintDialog`, passing the window reference for the parent window in the `documentWindow` parameter and a universal procedure pointer to an application-defined (callback) function in the `sheetDoneProc` parameter. The callback function should perform the actions required immediately following dismissal of the dialog, and should be declared like this:

```
void  myPageSetupSheetDoneFunction(PMPrintSession printSession, WindowRef documentWindow,
                                   Boolean accepted);
```

The `accepted` formal parameter will be set to `true` if the Print/OK push button is clicked and to `false` if the Cancel push button is clicked.

The functions `NewPMSheetDoneUPP` and `DisposePMSheetDoneUPP` create and dispose of the universal procedure pointers.

Saving and Retrieving a Page Format Object

As previously stated, the only information you should preserve each time the user prints a document should be that obtained via the Page Setup dialog. Ordinarily, therefore, you will want your application to save the flattened `PMPageFormat` object associated with a specific document in either the data fork or the resource fork of that document's file when you save the document itself.

You can store additional data inside the `PMPageFormat` object before it is flattened and saved by calling `PMSetPageFormatExtendedData`, whose parameters include a unique code identifying the data, the size of the data, and a pointer to the data. When the flattened `PMPageFormat` object is retrieved and unflattened, you can obtain this data by calling `PMGetPageFormatExtendedData`.

Saving a flattened `PMPageFormat` object to the resource fork of a file, and retrieving the flattened object from the file, is demonstrated in the demonstration program associated with Chapter 19.

Printing from the Finder-Mac OS 8/9

Users generally print documents that are open on the screen one at a time while the application that created the document is running. However, on Mac OS 8/9, users can also print one or more documents from the Finder by selecting the documents and choosing Print… from the Finder's File menu. This causes the Finder to launch the application and pass it a required Apple event (the Print Documents event) indicating the documents to be printed. In response to a Print Documents event, your application should

- Use saved or default page setup settings instead of displaying the Page Setup dialog
- Display the Print dialog once only, and use `PMCopyPrintSettings` to apply the information specified by the user to all of the selected documents
- Remain open unless and until the Finder sends it a Quit Application event

Main Carbon Printing Manager Constants, Data Types and Functions

Constants

Unwanted Data

```
KPMNoData               = NULL
KPMDontWantSize         = NULL
KPMDontWantData         = NULL
KPMDontWantBoolean      = NULL
KPMNoPrintSettings      = NULL
kPMNoPageFormat         = NULL
kPMNoReference          = NULL
```

Page Orientation

```
kPMPortrait             = 1
kPMLandscape            = 2
kPMReversePortrait      = 3
kPMReverseLandscape     = 4
```

User Cancelled

```
kPMCancel               = 128
```

For PMSetPageRange and PMSetLastPage

```
kPMPrintAllPages        = -1
```

Result Codes

```
kPMNoError              = 0
kPMGeneralError         = -30870
kPMOutOfScope           = -30871
kPMInvalidParameter     = paramErr
kPMNoDefaultPrinter     = -30872
kPMNotImplemented       = -30873
kPMNoSuchEntry          = -30874
kPMInvalidPrintSettings = -30875
kPMInvalidPageFormat    = -30876
kPMValueOutOfRange      = -30877
kPMLockIgnored          = -30878
kPMInvalidPrintSession  = -30879
kPMInvalidPrinter       = -30880
kPMObjectInUse          = -30881
```

Data Types

Opaque Types

```
typedef struct OpaquePMPrintSession*  PMPrintSession;
typedef struct OpaquePMPageFormat*    PMPageFormat;
typedef struct OpaquePMPrintSettings* PMPrintSettings;
typedef struct OpaquePMPrintContext*  PMPrintContext;
typedef struct OpaquePMDialog*        PMDialog;
```

PMRect

```
struct PMRect
{
  double top;
  double left;
  double right;
  double bottom;
}
```

PMResolution

```
struct PMResolution
{
  double hRes;
  double vRes;
}
```

Functions

Managing Printing Objects

```
OSStatus  PMRetain(PMObject object);
OSStatus  PMRelease(PMObject object);
```

Print Loop

```
OSStatus  PMCreateSession(PMPrintSession *printSession);
OSStatus  PMSessionBeginDocument(PMPrintSession printSession,PMPrintSettings printSettings,
          PMPageFormat pageFormat);
OSStatus  PMSessionEndDocument(PMPrintSession printSession);
OSStatus  PMSessionBeginPage(PMPrintSession printSession,PMPageFormat pageFormat,
          const PMRect *pageFrame);
OSStatus  PMSessionEndPage(PMPrintSession printSession);
OSStatus  PMSessionGetGraphicsContext(PMPrintSession printSession,
          CFStringRef graphicsContextType,void **graphicsContext);
```

Page Format and Print Settings Objects

```
OSStatus  PMCreatePageFormat(PMPageFormat *pageFormat);
OSStatus  PMCreatePrintSettings(PMPrintSettings *printSettings);
OSStatus  PMSessionDefaultPageFormat(PMPrintSession printSession,PMPageFormat pageFormat);
OSStatus  PMSessionDefaultPrintSettings(PMPrintSession printSession,
          PMPrintSettings printSettings);
OSStatus  PMSessionValidatePageFormat(PMPrintSession printSession,PMPageFormat pageFormat,
          Boolean *result);
OSStatus  PMSessionValidatePrintSettings(PMPrintSession printSession,
          PMPrintSettings printSettings,Boolean *result);
OSStatus  PMCopyPageFormat(PMPageFormat formatSrc,PMPageFormat formatDest);
OSStatus  PMCopyPrintSettings(PMPrintSettings settingSrc,PMPrintSettings settingDest);
OSStatus  PMFlattenPageFormat(PMPageFormat pageFormat,Handle *flatFormat);
OSStatus  PMUnflattenPageFormat(Handle flatFormat,PMPageFormat *pageFormat);
OSStatus  PMFlattenPrintSettings(PMPrintSettings printSettings,Handle *flatSetting);
OSStatus  PMUnflattenPrintSettings(Handle flatSetting,PMPrintSettings *printSettings);
```

Displaying the Page Setup and Print Dialogs

```
OSStatus  PMSessionPageSetupDialog(PMPrintSession printSession,PMPageFormat pageFormat,
          Boolean *accepted);
OSStatus  PMSessionPrintDialog(PMPrintSession printSession,PMPrintSettings printSettings,
          PMPageFormat constPageFormat,Boolean* accepted);
OSStatus  PMSessionUseSheets(PMPrintSession printSession,WindowRef documentWindow,
          PMSheetDoneUPP sheetDoneProc);
```

Customizing Page Setup and Print Dialogs

```
OSStatus  PMSessionPageSetupDialogInit(PMPrintSession printSession,PMPageFormat pageFormat,
          PMDialog *newDialog);
OSStatus  PMSessionPrintDialogInit(PMPrintSession printSession,PMPrintSettings
printSettings,
          PMPageFormat constPageFormat,PMDialog *newDialog);
OSStatus  PMSessionPageSetupDialogMain(PMPrintSession printSession,PMPageFormat pageFormat,
          Boolean *accepted,PMPageSetupDialogInitUPP myInitProc);
OSStatus  PMSessionPrintDialogMain(PMPrintSession printSession,PMPrintSettings
printSettings,
          PMPageFormat constPageFormat,Boolean *accepted,PMPrintDialogInitUPP myInitProc);
OSStatus  PMSetModalFilterProc(PMDialog pmDialog,ModalFilterUPP filterProc);
OSStatus  PMSetItemProc(PMDialog pmDialog,PMItemUPP itemProc);
OSStatus  PMGetItemProc(PMDialog pmDialog,PMItemUPP *itemProc);
OSStatus  PMGetDialogPtr (PMDialog pmDialog, DialogRef *theDialog);
```

Getting and Setting Page Setup Information

```
OSStatus  PMGetAdjustedPaperRect(PMPageFormat pageFormat,PMRect *paperRect);
OSStatus  PMGetAdjustedPageRect(PMPageFormat pageFormat,PMRect *pageRect);
OSStatus  PMGetUnadjustedPaperRect(PMPageFormat pageFormat,PMRect *paperSize);
OSStatus  PMSetUnadjustedPaperRect(PMPageFormat pageFormat,const PMRect paperSize);
OSStatus  PMGetUnadjustedPageRect(PMPageFormat pageFormat,PMRect *pageSize);
OSStatus  PMGetResolution(PMPageFormat pageFormat,PMResolution *res);
OSStatus  PMSetResolution(PMPageFormat pageFormat,const PMResolution res);
OSStatus  PMSetOrientation(PMPageFormat pageFormat,PMOrientation orientation,Boolean lock);
OSStatus  PMGetOrientation(PMPageFormat pageFormat,PMOrientation *orientation);
OSStatus  PMGetScale(PMPageFormat pageFormat,double *scale);
OSStatus  PMSetScale(PMPageFormat pageFormat,double scale);
OSStatus  PMGetPageFormatExtendedData(PMPageFormat pageFormat,OSType dataID,UInt32 *size,
          void *extendedData);
OSStatus  PMSetPageFormatExtendedData(PMPageFormat pageFormat,OSType dataID,UInt32 size,
          void *extendedData);
```

Getting and Setting Printing Information

```
OSStatus  PMGetFirstPage(PMPrintSettings printSettings,UInt32 *first);
OSStatus  PMSetFirstPage(PMPrintSettings printSettings,UInt32 first,Boolean lock);
OSStatus  PMGetLastPage(PMPrintSettings printSettings,UInt32 *last);
OSStatus  PMSetLastPage(PMPrintSettings printSettings,UInt32 last,Boolean lock);
OSStatus  PMGetPageRange(PMPrintSettings printSettings,UInt32 *minPage,UInt32 *maxPage);
OSStatus  PMSetPageRange(PMPrintSettings printSettings,UInt32 minPage,UInt32 maxPage);
OSStatus  PMGetCopies(PMPrintSettings printSettings,UInt32 *copies);
OSStatus  PMSetCopies(PMPrintSettings printSettings,UInt32 copies,Boolean lock);
OSStatus  PMGetPrintSettingsExtendedData(PMPrintSettings printSettings,OSType dataID,
          UInt32 *size,void *extendedData);
OSStatus  PMSetPrintSettingsExtendedData(PMPrintSettings printSettings,OSType dataID,
          UInt32 size,void *extendedData);
```

Creating and Disposing of Universal Procedure Pointers

```
PMPageSetupDialogInitUPP NewPMPageSetupDialogInitUPP(PMPageSetupDialogInitProcPtr
userRoutine);
PMPrintDialogInitUPP     NewPMPrintDialogInitUPP(PMPrintDialogInitProcPtr userRoutine);
PMItemUPP                NewPMItemUPP(PMItemProcPtr userRoutine);
PMIdleUPP                NewPMIdleUPP(PMIdleProcPtr userRoutine);
PMSheetDoneUPP           NewPMSheetDoneUPP(PMSheetDoneProcPtr userRoutine);
void      DisposePMPageSetupDialogInitUPP(PMPageSetupDialogInitUPP userUPP);
Void      DisposePMPrintDialogInitUPP    (PMPrintDialogInitUPP userUPP);
void      DisposePMItemUPP(PMItemUPP userUPP);
void      DisposePMIdleUPP(PMIdleUPP userUPP);
void      DisposePMSheetDoneUPP(PMSheetDoneUPP userUPP);
```

Errors

```
OSStatus  PMSessionError(PMPrintSession printSession);
```

Application-Defined (Callback) Functions

```
void      myPMDialogSheetDoneFunction(PMPrintSession printSession, WindowRef documentWindow,
                                      Boolean accepted);
```

Demonstration Program CarbonPrinting Listing

```
//
/*****************************************************************************
// CarbonPrinting.h                                                  CLASSIC EVENT
MODEL
//
*****************************************************************************/
//
// This program:
//
// •  Demonstrates printing using the Carbon Printing Manager session functions.
//
// •  Opens two windows. The first window is a document window in which is displayed the
//    document to be printed. The second window displays some printing-related information
//    obtained from PMPageFormat and PMPrintSettings objects.
//
// •  Customizes the Pring dialog by adding a pop-up menu button, three radio buttons, a
//    checkbox, and a group box.
//
// •  Allows the user to print a document containing a picture and text, with the text
//    being printed in the font and font size, and with the fractional widths setting,
//    specified using the items added to the Print dialog.
//
// The customizing of the Print dialog uses the AppendDITL method. Accordingly, on Mac OS X,
// the dialogs are application-modal and are not displayed as window-modal sheet dialogs.
//
// The program utilizes the following resources:
//
// •  A 'plst' resource.
//
// •  'MBAR' resource and associated 'MENU' resources (preload, non-purgeable).
//
// •  Two 'WIND' resources (purgeable).
//
// •  A 'TEXT' resource (purgeable) used for printing.
//
// •  A 'PICT' resource (non-purgeable) used for printing.
//
// •  'CNTL' resources (purgeable) for controls added to the Print dialog box.
//
// •  A 'DITL' resource (purgeable) specifying the items to be appended to the Print dialog
//    box.
//
// •  A 'MENU' resource (preload, non-purgeable) for the pop-up menu button.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
//
/*****************************************************************************/

// ..........................................................................
includes

#include <Carbon.h>

// ..........................................................................
defines

#define rMenubar                128
#define mAppleApplication       128
```

```
#define  iAbout               1
#define mFile                 129
#define  iPageSetup           9
#define  iPrint               10
#define  iQuit                12
#define mFont                 131
#define rDocWindow            128
#define rInfoWindow           129
#define rText                 128
#define rPicture              128
#define rPrintDialogAppendDITL 128
#define  iPopupButton         1
#define  iRadioButton10pt     2
#define  iRadioButton12pt     3
#define  iRadioButton14pt     4
#define  iCheckboxFracWidths  5
#define kMargin               90
#define MAX_UINT32            0xFFFFFFFF
#define MIN(a,b)              ((a) < (b) ? (a) : (b))
// ................................................................................
typedefs

typedef struct
{
  PMPrintSession  printSession;
  PMPageFormat    pageFormat;
  PMPrintSettings printSettings;
  TEHandle        editTextStrucHdl;
  PicHandle       pictureHdl;
}  docStructure, *docStructurePtr, **docStructureHdl;

// ..................................................................... function
prototypes

void      main                          (void);
void      doPreliminaries               (void);
OSErr      quitAppEventHandler          (AppleEvent *,AppleEvent *,SInt32);
void      doGetDocument                 (void);
void      doEvents                      (EventRecord *);
void      doUpdate                      (EventRecord *);
void      doUpdateDocumentWindow        (WindowRef);
void      doMenuChoice                  (SInt32);
OSStatus  doCreateOrValidatePageFormat  (WindowRef);
OSStatus  doPageSetUpDialog             (WindowRef);
OSStatus  doPrintSettingsDialog         (WindowRef);
OSStatus  doPrinting                    (WindowRef);
SInt16    doCalcNumberOfPages           (WindowRef,Rect);
void      doDrawPage                    (WindowRef,Rect,SInt16,SInt16);
void      doDrawPrintInfo               (void);
void      doDrawRectStrings             (Str255,SInt16,SInt16,Str255,SInt16,SInt16,Str255);
void      doErrorAlert                  (OSStatus);
void      doConcatPStrings              (Str255,Str255);

void      initializationFunction        (PMPrintSettings, PMDialog *);
void      itemEvaluationFunction        (DialogPtr,SInt16);
Boolean   eventFilter                   (DialogPtr,EventRecord *,SInt16 *);

//
******************************************************************************************
// CarbonPrinting.c
//
******************************************************************************************

// ..............................................................................
includes
```

```
#include "CarbonPrinting.h"

// ..................................................................... global
variables

Boolean         gRunningOnX = false;
WindowRef       gDocumentWindowRef, gPrintInfoWindowRef;
SInt16          gFontNumber;
SInt16          gFontSize;
Boolean         gDone;
PMItemUPP       gNewItemEvaluateFunctionUPP;
ModalFilterUPP  gEventFilterUPP;
PMPrintSettings gPrintSettings = kPMNoPrintSettings;
PMDialog        gPMDialog;
UInt32          gFirstPage, gLastPage, gCopies;
Boolean         gDrawPrintSettingsStuff;

// ********************************************************************************
main

void  main(void)
{
  MenuBarHandle   menubarHdl;
  SInt32          response;
  MenuRef         menuRef;
  docStructureHdl docStrucHdl;
  SInt16          fontNum;
  RGBColor        whiteColor = {  0xFFFF, 0xFFFF, 0xFFFF } ;
  RGBColor        blueColor  = {  0x4444, 0x4444, 0x9999 } ;
  Rect            portRect;
  EventRecord     eventStructure;
  Boolean         gotEvent;

  // ..................................................................... do
prelimiaries

  doPreliminaries();

  // ..................................................................... set up menu bar and
menus

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    ExitToShell();
  SetMenuBar(menubarHdl);
  DrawMenuBar();

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
    }

    gRunningOnX = true;
  }

  // ..................................... open document window and attach document
structure

  if(!(gDocumentWindowRef = GetNewCWindow(rDocWindow,NULL,(WindowRef)-1)))
    ExitToShell();
```

```
  SetPortWindowPort(gDocumentWindowRef);
  GetFNum("\pGeneva",&fontNum);
  TextFont(fontNum);
  TextSize(10);
  gFontNumber = fontNum;
  gFontSize = 10;
  if(!(docStrucHdl = (docStructureHdl) NewHandle(sizeof(docStructure))))
    ExitToShell();
  SetWRefCon(gDocumentWindowRef,(SInt32) docStrucHdl);

  (*docStrucHdl)->printSession  = NULL;
  (*docStrucHdl)->pageFormat    = kPMNoPageFormat;
  (*docStrucHdl)->printSettings = kPMNoPrintSettings;

  // ..................................................... open printing information
window

  if(!(gPrintInfoWindowRef = GetNewCWindow(rInfoWindow,NULL,(WindowRef)-1)))
    ExitToShell();

  SetPortWindowPort(gPrintInfoWindowRef);
  TextFont(fontNum);
  TextSize(10);
  RGBForeColor(&whiteColor);
  RGBBackColor(&blueColor);
  GetWindowPortBounds(gPrintInfoWindowRef,&portRect);
  EraseRect(&portRect);

  // ................................................. load and display simulated
document

  doGetDocument();

  // ............................................................................. event
loop

  gDone = false;

  while(!gDone)
  {
    gotEvent = WaitNextEvent(everyEvent,&eventStructure,MAX_UINT32,NULL);
    if(gotEvent)
      doEvents(&eventStructure);
  }
}

// ************************************************************************
doPreliminaries

void  doPreliminaries(void)
{
  OSErr osError;

  MoreMasterPointers(128);
  InitCursor();
  FlushEvents(everyEvent,0);

  osError = AEInstallEventHandler(kCoreEventClass,kAEQuitApplication,
                            NewAEEventHandlerUPP((AEEventHandlerProcPtr)
quitAppEventHandler),
                            0L,false);
  if(osError != noErr)
    ExitToShell();
}
// ************************************************************************
doQuitAppEvent
```

```
 OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  OSErr    osError;
  DescType returnedType;
  Size     actualSize;

  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,
NULL,0,
                              &actualSize);

  if(osError == errAEDescNotFound)
  {
    gDone = true;
    osError = noErr;
  }
  else if(osError == noErr)
    osError = errAEParamMissed;

  return osError;
}

// ************************************************************************
doGetDocument

void  doGetDocument(void)
{
  docStructureHdl docStrucHdl;
  Rect            portRect, destRect, viewRect;
  Handle          textHdl;

  SetPortWindowPort(gDocumentWindowRef);

  docStrucHdl = (docStructureHdl) GetWRefCon(gDocumentWindowRef);

  GetWindowPortBounds(gDocumentWindowRef,&portRect);
  destRect = portRect;
  InsetRect(&destRect,4,4);
  destRect.bottom +=4;
  viewRect = destRect;
  (*docStrucHdl)->editTextStrucHdl = TENew(&destRect,&viewRect);

  textHdl = GetResource('TEXT',rText);
  if(textHdl == NULL)
    ExitToShell();

  HLock(textHdl);
  TEInsert(*textHdl,GetHandleSize(textHdl),(*docStrucHdl)->editTextStrucHdl);
  HUnlock(textHdl);

  ReleaseResource(textHdl);

  (*docStrucHdl)->pictureHdl = GetPicture(rPicture);
  if((*docStrucHdl)->pictureHdl == NULL)
    ExitToShell();
  InvalWindowRect(gDocumentWindowRef,&portRect);
}

// ************************************************************************
doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
  WindowRef      windowRef;
  WindowPartCode partCode;
```

```
  windowRef = (WindowRef) eventStrucPtr->message;

  switch(eventStrucPtr->what)
  {
    case kHighLevelEvent:
      AEProcessAppleEvent(eventStrucPtr);
      break;

    case mouseDown:
      partCode = FindWindow(eventStrucPtr->where,&windowRef);
      switch(partCode)
      {
        case inMenuBar:
          doMenuChoice(MenuSelect(eventStrucPtr->where));
          break;

        case inContent:
          if(windowRef != FrontWindow())
            SelectWindow(windowRef);
          break;

        case inDrag:
          DragWindow(windowRef,eventStrucPtr->where,NULL);
          break;
      }
      break;

    case keyDown:
      if((eventStrucPtr->modifiers & cmdKey) != 0)
        doMenuChoice(MenuEvent(eventStrucPtr));
      break;

    case updateEvt:
      doUpdate(eventStrucPtr);
      break;
  }
}

// *****************************************************************************
doUpdate

void  doUpdate(EventRecord *eventStrucPtr)
{
  WindowRef windowRef;
  GrafPtr   oldPort;
  Rect      portRect;
  windowRef = (WindowRef) eventStrucPtr->message;

  GetPort(&oldPort);

  BeginUpdate(windowRef);

  if(windowRef == gDocumentWindowRef)
    doUpdateDocumentWindow(windowRef);
  else if(windowRef == gPrintInfoWindowRef)
  {
    SetPortWindowPort(gPrintInfoWindowRef);
    GetWindowPortBounds(gPrintInfoWindowRef,&portRect);
    EraseRect(&portRect);
    doDrawPrintInfo();
  }

  EndUpdate(windowRef);

  SetPort(oldPort);
}
```

```
// ************************************************************************
doUpdateDocumentWindow

void  doUpdateDocumentWindow(WindowRef windowRef)
{
  Rect            portRect, pictureRect, savedDestRect;
  docStructureHdl docStrucHdl;
  SInt16          savedFontNum, savedFontSize, savedLineHeight, fontNum;

  SetPortWindowPort(windowRef);
  GetWindowPortBounds(gDocumentWindowRef,&portRect);
  docStrucHdl = (docStructureHdl) GetWRefCon(windowRef);

  savedDestRect = (*(*docStrucHdl)->editTextStrucHdl)->destRect;
  savedFontNum = (*(*docStrucHdl)->editTextStrucHdl)->txFont;
  savedFontSize = (*(*docStrucHdl)->editTextStrucHdl)->txSize;
  savedLineHeight = (*(*docStrucHdl)->editTextStrucHdl)->lineHeight;

  (*(*docStrucHdl)->editTextStrucHdl)->destRect = portRect;
  InsetRect(&(*(*docStrucHdl)->editTextStrucHdl)->destRect,4,4);
  (*(*docStrucHdl)->editTextStrucHdl)->destRect.bottom += 4;
  GetFNum("\pGeneva",&fontNum);
  (*(*docStrucHdl)->editTextStrucHdl)->txFont = fontNum;
  (*(*docStrucHdl)->editTextStrucHdl)->txSize = 10;
  (*(*docStrucHdl)->editTextStrucHdl)->lineHeight = 13;
  TECalText((*docStrucHdl)->editTextStrucHdl);
  TEUpdate(&portRect,(*docStrucHdl)->editTextStrucHdl);

  (*(*docStrucHdl)->editTextStrucHdl)->destRect = savedDestRect;
  (*(*docStrucHdl)->editTextStrucHdl)->txFont = savedFontNum;
  (*(*docStrucHdl)->editTextStrucHdl)->txFont = savedFontSize;
  (*(*docStrucHdl)->editTextStrucHdl)->lineHeight = savedLineHeight;
  SetRect(&pictureRect,2,2,180,134);
  DrawPicture((*docStrucHdl)->pictureHdl,&pictureRect);
}

// ************************************************************************
doMenuChoice

void  doMenuChoice(SInt32 menuChoice)
{
  MenuID        menuID;
  MenuItemIndex menuItem;
  OSStatus      osStatus;
  Rect          portRect;

  menuID = HiWord(menuChoice);
  menuItem = LoWord(menuChoice);

  if(menuID == 0)
    return;

  switch(menuID)
  {
    case mAppleApplication:
      if(menuItem == iAbout)
        SysBeep(10);
      break;

    case mFile:
      if(menuItem == iPageSetup)
      {
        osStatus = doPageSetUpDialog(gDocumentWindowRef);
        if(osStatus != kPMNoError)
          doErrorAlert(osStatus);
```

```
      }

      if(menuItem == iPrint)
      {
        osStatus = doPrintSettingsDialog(gDocumentWindowRef);
        if(osStatus == kPMNoError)
        {
          osStatus = doPrinting(gDocumentWindowRef);
          if(osStatus != kPMNoError)
            doErrorAlert(osStatus);
        }
        else if(osStatus != kPMCancel)
          doErrorAlert(osStatus);
      }

      GetWindowPortBounds(gPrintInfoWindowRef,&portRect);
      InvalWindowRect(gPrintInfoWindowRef,&portRect);

      if(menuItem == iQuit)
        gDone = true;
      break;
  }

  HiliteMenu(0);
}

// ************************************************************
doCreateOrValidatePageFormat

OSStatus  doCreateOrValidatePageFormat (WindowRef windowRef)
{
  docStructureHdl docStrucHdl;
  OSStatus        osStatus     = kPMNoError;
  PMPrintSession  printSession = NULL;
  PMPageFormat    pageFormat   = kPMNoPageFormat;

  docStrucHdl = (docStructureHdl) GetWRefCon(windowRef);
  HLock((Handle) docStrucHdl);

  // ..................................................... create printing
session

  osStatus = PMCreateSession(&printSession);

  // .......... if necessary, create and store page format object, otherwise validate
existing

  if(osStatus == noErr)
  {
    if((*docStrucHdl)->pageFormat == kPMNoPageFormat)
    {
      osStatus = PMCreatePageFormat(&pageFormat);

      if((osStatus == kPMNoError) && (pageFormat != kPMNoPageFormat))
      {
        osStatus = PMSessionDefaultPageFormat(printSession,pageFormat);

        if(osStatus == kPMNoError)
          (*docStrucHdl)->pageFormat = pageFormat;
      }
      else
      {
        if(osStatus == kPMNoError)
          osStatus = kPMGeneralError;
      }
    }
```

```
    else
    {
      osStatus = PMSessionValidatePageFormat(printSession,(*docStrucHdl)->pageFormat,
                                             kPMDontWantBoolean);
    }
  }
  // ..................................... store printing session, or clean up if
error

  if(osStatus == kPMNoError)
    (*docStrucHdl)->printSession = printSession;
  else
  {
    if(pageFormat != kPMNoPageFormat)
      PMRelease(pageFormat);
    if(printSession != NULL)
      PMRelease(printSession);
  }

  HUnlock((Handle) docStrucHdl);

  return osStatus;
}

// ************************************************************************
doPageSetUpDialog

OSStatus  doPageSetUpDialog(WindowRef windowRef)
{
  OSStatus        osStatus = kPMNoError;
  docStructureHdl docStrucHdl;
  Boolean         userClickedOKButton;

  osStatus = doCreateOrValidatePageFormat (windowRef);
  if(osStatus != kPMNoError)
    return osStatus;

  docStrucHdl = (docStructureHdl) GetWRefCon(windowRef);
  HLock((Handle) docStrucHdl);

  osStatus = PMSessionPageSetupDialog((*docStrucHdl)->printSession,(*docStrucHdl)-
>pageFormat,
                                      &userClickedOKButton);

  if((*docStrucHdl)->printSession != NULL)
  {
    PMRelease((*docStrucHdl)->printSession);
    (*docStrucHdl)->printSession = NULL;
  }

  HUnlock((Handle) docStrucHdl);
  gDrawPrintSettingsStuff = false;

  return osStatus;
}

// ************************************************************************
doPrintSettingsDialog

OSStatus  doPrintSettingsDialog(WindowRef windowRef)
{
  OSStatus             osStatus      = kPMNoError;
  docStructureHdl      docStrucHdl;
  PMPrintSettings      printSettings = kPMNoPrintSettings;
  CFStringRef          stringRef;
  PMPrintDialogInitUPP initializationFunctionUPP;
```

```
  Boolean             userClickedPrintButton;

  osStatus = doCreateOrValidatePageFormat (windowRef);
  if(osStatus != kPMNoError)
    return osStatus;

  docStrucHdl = (docStructureHdl) GetWRefCon(windowRef);
  HLock((Handle) docStrucHdl);

  osStatus = PMCreatePrintSettings(&printSettings);
  if((osStatus == kPMNoError) && (printSettings != kPMNoPrintSettings))
  {
    osStatus = CopyWindowTitleAsCFString(windowRef,&stringRef);
    if(osStatus == noErr)
    {
      osStatus = PMSetJobNameCFString(printSettings,stringRef);
      CFRelease(stringRef);
    }

    if(osStatus == noErr)
      osStatus = PMSessionDefaultPrintSettings((*docStrucHdl)->printSession,printSettings);

    if(osStatus == kPMNoError)
    {
      (*docStrucHdl)->printSettings = printSettings;
      printSettings = kPMNoPrintSettings;
    }
  }

  if(osStatus == kPMNoError)
  {
    if(gRunningOnX)
    {
      PMSetPageRange((*docStrucHdl)->printSettings,1,kPMPrintAllPages);
      PMSetFirstPage((*docStrucHdl)->printSettings,1,false);
      PMSetLastPage((*docStrucHdl)->printSettings,9999,false);
    }
  }

  if(osStatus == kPMNoError)
  {
    initializationFunctionUPP   = NewPMPrintDialogInitUPP((PMPrintDialogInitProcPtr)
                                                          initializationFunction);
    gNewItemEvaluateFunctionUPP = NewPMItemUPP((PMItemProcPtr) itemEvaluationFunction);
    gEventFilterUPP             = NewModalFilterUPP((ModalFilterProcPtr) eventFilter);

    osStatus = PMSessionPrintDialogInit((*docStrucHdl)->printSession,
                                        (*docStrucHdl)->printSettings,
                                        (*docStrucHdl)->pageFormat,&gPMDialog);
    if(osStatus == kPMNoError)
      osStatus = PMSessionPrintDialogMain((*docStrucHdl)->printSession,
                                          (*docStrucHdl)->printSettings,
                                          (*docStrucHdl)->pageFormat,
                                          &userClickedPrintButton,
initializationFunctionUPP);

    if(osStatus == kPMNoError && !userClickedPrintButton)
      osStatus = kPMCancel;

    DisposePMPrintDialogInitUPP(initializationFunctionUPP);
    DisposePMItemUPP(gNewItemEvaluateFunctionUPP);
    DisposeModalFilterUPP(gEventFilterUPP);
  }

  if(osStatus != kPMNoError || osStatus == kPMCancel)
  {
```

```
    if((*docStrucHdl)->printSettings != kPMNoPrintSettings)
    {
      PMRelease((*docStrucHdl)->printSettings);
      (*docStrucHdl)->printSettings = kPMNoPrintSettings;
    }
    if((*docStrucHdl)->printSession != NULL)
    {
      PMRelease((*docStrucHdl)->printSession);
      (*docStrucHdl)->printSession = NULL;
    }
  }

  HUnlock((Handle) docStrucHdl);
  gDrawPrintSettingsStuff = userClickedPrintButton;

  return osStatus;
}

// ****************************************************************************
doPrinting

OSStatus  doPrinting(WindowRef windowRef)
{
  docStructureHdl docStrucHdl;
  OSStatus        osStatus = kPMNoError;
  UInt32          firstPage, lastPage, numberOfPages;
  PMRect          adjustedPageRect;
  Rect            pageRect;
  SInt16          page;
  GrafPtr         oldPort, currentPort, printingPort;

  GetPort(&oldPort);
  docStrucHdl = (docStructureHdl) GetWRefCon(windowRef);
  HLock((Handle) docStrucHdl);
  // ............................get first and last page to print as set by user in Print
dialog

  osStatus = PMGetFirstPage((*docStrucHdl)->printSettings,&firstPage);

  if(osStatus == kPMNoError)
    osStatus = PMGetLastPage((*docStrucHdl)->printSettings,&lastPage);

  // for demo purposes, store first, last page and copies for Some Printing Information
window

  gFirstPage = firstPage;
  gLastPage = lastPage;
  PMGetCopies((*docStrucHdl)->printSettings,&gCopies);

  // ..................get actual number of pages in document and, if necessary, adjust
last page

  if(osStatus == kPMNoError)
    osStatus = PMGetAdjustedPageRect((*docStrucHdl)->pageFormat,&adjustedPageRect);

  if(osStatus == kPMNoError)
  {
    pageRect.top    = adjustedPageRect.top;
    pageRect.left   = adjustedPageRect.left;
    pageRect.bottom = adjustedPageRect.bottom;
    pageRect.right  = adjustedPageRect.right;

    numberOfPages = doCalcNumberOfPages(windowRef,pageRect);

    if(numberOfPages < lastPage)
      lastPage = numberOfPages;
```

```
  }

  // ................................. for Mac OS X, set first and last page for
progress dialog

  if(gRunningOnX)
  {
    if(osStatus == kPMNoError)
      osStatus = PMSetFirstPage((*docStrucHdl)->printSettings,firstPage,false);

    if(osStatus == kPMNoError)
      osStatus = PMSetLastPage((*docStrucHdl)->printSettings,lastPage,false);
  }

  // ..........................................................................
printing loop

  if(osStatus == kPMNoError)
  {
    osStatus = PMSessionBeginDocument((*docStrucHdl)->printSession,
                                      (*docStrucHdl)->printSettings,
                                      (*docStrucHdl)->pageFormat);
    if(osStatus == kPMNoError)
    {
      page = 1;
      while((page <= lastPage) && (osStatus == kPMNoError) &&
            (PMSessionError((*docStrucHdl)->printSession) == kPMNoError))
      {
        osStatus = PMSessionBeginPage((*docStrucHdl)->printSession,
                                      (*docStrucHdl)->pageFormat,NULL);
        if(osStatus != kPMNoError)
          break;

        GetPort(&currentPort);

        osStatus = PMSessionGetGraphicsContext((*docStrucHdl)->printSession,
                                               kPMGraphicsContextQuickdraw,
                                               (void **) &printingPort);
        if(osStatus == kPMNoError)
        {
          SetPort(printingPort);
          doDrawPage(windowRef,pageRect,page,numberOfPages);
          SetPort(currentPort);
        }

        osStatus = PMSessionEndPage((*docStrucHdl)->printSession);
        if(osStatus != kPMNoError)
          break;

        page++;
      }
    }

    PMSessionEndDocument((*docStrucHdl)->printSession);
  }

  // ..........................................................................
clean up

  if((*docStrucHdl)->printSettings != kPMNoPrintSettings)
  {
    PMRelease((*docStrucHdl)->printSettings);
    (*docStrucHdl)->printSettings = kPMNoPrintSettings;
  }
  if((*docStrucHdl)->printSession != NULL)
  {
```

```
    PMRelease((*docStrucHdl)->printSession);
    (*docStrucHdl)->printSession = NULL;
  }

  HUnlock((Handle) docStrucHdl);
  SetPort(oldPort);

  return osStatus;
}
// *****************************************************************
doCalcNumberOfPages

SInt16  doCalcNumberOfPages(WindowRef windowRef,Rect pageRect)
{
  docStructureHdl docStrucHdl;
  FontInfo        fontInfo;
  Rect            destRect;
  SInt16          heightDestRect, linesPerPage, numberOfPages;

  docStrucHdl = (docStructureHdl) GetWRefCon(windowRef);

  TextFont(gFontNumber);
  TextSize(gFontSize);
  GetFontInfo(&fontInfo);

  (*(*docStrucHdl)->editTextStrucHdl)->txFont = gFontNumber;
  (*(*docStrucHdl)->editTextStrucHdl)->txSize = gFontSize;
  (*(*docStrucHdl)->editTextStrucHdl)->lineHeight = fontInfo.ascent + fontInfo.descent
                                                    + fontInfo.leading;

  SetRect(&destRect,pageRect.left + kMargin,pageRect.top + (kMargin * 1.5),
          pageRect.right - kMargin,pageRect.bottom - (kMargin * 1.5));

  (*(*docStrucHdl)->editTextStrucHdl)->destRect = destRect;

  TECalText((*docStrucHdl)->editTextStrucHdl);

  heightDestRect = destRect.bottom - destRect.top;
  linesPerPage = heightDestRect / (*(*docStrucHdl)->editTextStrucHdl)->lineHeight;
  numberOfPages = ((*(*docStrucHdl)->editTextStrucHdl)->nLines / linesPerPage) + 1;

  return(numberOfPages);
}

// *****************************************************************************
doDrawPage

void  doDrawPage(WindowRef windowRef,Rect pageRect,SInt16 pageNumber,SInt16 numberOfpages)
{
  docStructureHdl docStrucHdl;
  TEHandle        docEditTextStrucHdl, pageEditTextStrucHdl;
  PicHandle       pictureHdl;
  Rect            destRect, pictureRect;
  SInt16          heightDestRect, linesPerPage, numberOfLines;
  Handle          textHdl;
  SInt32          startOffset, endOffset;
  Str255          theString;

  TextFont(gFontNumber);
  TextSize(gFontSize);

  docStrucHdl = (docStructureHdl) GetWRefCon(windowRef);
  docEditTextStrucHdl  = (*docStrucHdl)->editTextStrucHdl;
  pictureHdl = (*docStrucHdl)->pictureHdl;
  destRect = (*docEditTextStrucHdl)->destRect;
  heightDestRect = destRect.bottom - destRect.top;
```

```
  linesPerPage = heightDestRect / (*docEditTextStrucHdl)->lineHeight;

  numberOfLines = (*docEditTextStrucHdl)->nLines;
  startOffset = (*docEditTextStrucHdl)->lineStarts[(pageNumber - 1) * linesPerPage];
  if(pageNumber == numberOfpages)
    endOffset = (*docEditTextStrucHdl)->lineStarts[numberOfLines];
  else
    endOffset = (*docEditTextStrucHdl)->lineStarts[pageNumber * linesPerPage];

  pageEditTextStrucHdl = TENew(&destRect,&destRect);
  textHdl = (*docEditTextStrucHdl)->hText;
  HLock(textHdl);
  TEInsert(*textHdl + startOffset,endOffset - startOffset,pageEditTextStrucHdl);
  TEDispose(pageEditTextStrucHdl);

  if(pageNumber == 1)
  {
    SetRect(&pictureRect,destRect.left,destRect.top,
            destRect.left + ((*pictureHdl)->picFrame.right - (*pictureHdl)->picFrame.left),
            destRect.top + ((*pictureHdl)->picFrame.bottom - (*pictureHdl)->picFrame.top));
    DrawPicture(pictureHdl,&pictureRect);
  }

  MoveTo(destRect.left,pageRect.bottom - 25);
  NumToString((SInt32) pageNumber,theString);
  DrawString(theString);
}

// ************************************************************************
doDrawPrintInfo

void  doDrawPrintInfo(void)
{
  docStructureHdl docStrucHdl;
  OSStatus        osStatus = kPMNoError;
  PMRect          theRect;
  Str255          s2, s3;
  PMResolution    resolution;
  double          scale;
  PMOrientation   orientation;

  docStrucHdl = (docStructureHdl) GetWRefCon(gDocumentWindowRef);
  if((*docStrucHdl)->pageFormat == kPMNoPageFormat)
    return;

  HLock((Handle) docStrucHdl);

  MoveTo(20,25);
  TextFace(bold);
  DrawString("\pFrom PMPageFormat Object:");
  TextFace(normal);
  PMGetAdjustedPaperRect((*docStrucHdl)->pageFormat,&theRect);
  NumToString((SInt32) theRect.top,s2);
  NumToString((SInt32) theRect.left,s3);
  doDrawRectStrings("\pPaper Rectangle (top,left):",20,45,s2,190,45,s3);
  NumToString((SInt32) theRect.bottom,s2);
  NumToString((SInt32) theRect.right,s3);
  doDrawRectStrings("\pPaper Rectangle (bottom,right):",20,60,s2,190,60,s3);

  PMGetAdjustedPageRect((*docStrucHdl)->pageFormat,&theRect);
  NumToString((SInt32) theRect.top,s2);
  NumToString((SInt32) theRect.left,s3);
  doDrawRectStrings("\pPage Rectangle (top,left):",20,75,s2,190,75,s3);
  NumToString((SInt32) theRect.bottom,s2);
  NumToString((SInt32) theRect.right,s3);
  doDrawRectStrings("\pPage Rectangle (bottom,right):",20,90,s2,190,90,s3);
```

```
  PMGetResolution((*docStrucHdl)->pageFormat,&resolution);
  MoveTo(20,105);
  DrawString("\pDrawing Resolution (Vertical):");
  NumToString((SInt32) resolution.vRes,s2);
  MoveTo(190,105);
  DrawString(s2);
  DrawString("\p dpi");
  MoveTo(20,120);
  DrawString("\pDrawing Resolution (Horizontal):");
  NumToString((SInt32) resolution.hRes,s2);
  MoveTo(190,120);
  DrawString(s2);
  DrawString("\p dpi");

  PMGetScale((*docStrucHdl)->pageFormat,&scale);
  MoveTo(20,135);
  DrawString("\pScale:");
  NumToString((SInt32) scale,s2);
  MoveTo(190,135);
  DrawString(s2);
  DrawString("\p%");

  PMGetOrientation((*docStrucHdl)->pageFormat,&orientation);
  MoveTo(20,150);
  DrawString("\pPage Orientation:");
  MoveTo(190,150);
  if(orientation == kPMPortrait)
    DrawString("\pPortrait");
  else if(orientation == kPMLandscape)
    DrawString("\pLandscape");

  if(gDrawPrintSettingsStuff)
  {
    MoveTo(20,170);
    TextFace(bold);
    DrawString("\pFrom PMPrintSettings Object:");
    TextFace(normal);
    MoveTo(20,190);
    DrawString("\pFirst Page:");
    NumToString((SInt32) gFirstPage,s2);
    MoveTo(190,190);
    DrawString(s2);

    MoveTo(20,205);
    DrawString("\pLast Page:");
    NumToString((SInt32) gLastPage,s2);
    MoveTo(190,205);
    DrawString(s2);

    MoveTo(20,220);
    DrawString("\pNumber of Copies:");
    NumToString((SInt32) gCopies,s2);
    MoveTo(190,220);
    DrawString(s2);
  }

  HUnlock((Handle) docStrucHdl);
}

// ************************************************************************
doDrawRectStrings

void  doDrawRectStrings(Str255 s1,SInt16 x1,SInt16 y1,Str255 s2,SInt16 x2,SInt16 y2,Str255
s3)
{
```

```
  MoveTo(x1,y1);
  DrawString(s1);
  MoveTo(x2,y2);
  DrawString("\p(");
  DrawString(s2);
  DrawString("\p,");
  DrawString(s3);
  DrawString("\p)");
}

// ****************************************************************************
doErrorAlert

void  doErrorAlert(OSStatus errorType)
{
  Str255 theString, errorString = "\pCarbon Printing Manager Error ";
  SInt16 itemHit;

  NumToString((SInt32) errorType,theString);
  doConcatPStrings(errorString,theString);

  StandardAlert(kAlertCautionAlert,errorString,NULL,NULL,&itemHit);
}

// ****************************************************************************
doConcatPStrings

void  doConcatPStrings(Str255 targetString,Str255 appendString)
{
  SInt16  appendLength;
  appendLength = MIN(appendString[0],255 - targetString[0]);

  if(appendLength > 0)
  {
    BlockMoveData(appendString+1,targetString+targetString[0]+1,(SInt32) appendLength);
    targetString[0] += appendLength;
  }
}

//
**********************************************************************************************
// PrintDialogAppend.c
//
**********************************************************************************************

// ..........................................................................
includes

#include "CarbonPrinting.h"

// ..................................................................... global
variables

SInt32    gFirstAppendedItemNo;
PMItemUPP gOldItemEvaluateFunctionUPP;

extern PMDialog       gPMDialog;
extern PMItemUPP      gNewItemEvaluateFunctionUPP;
extern ModalFilterUPP gEventFilterUPP;
extern SInt16         gFontNumber;
extern SInt16         gFontSize;

// ****************************************************************
initializationFunction

void  initializationFunction(PMPrintSettings printSettings,PMDialog *pmDialog)
```

```
{
  OSStatus   osStatus = kPMNoError;
  DialogRef  dialogRef;
  Handle     ditlHdl;
  SInt16     numberOfExistingItems, numberOfMenuItems;
  MenuRef    menuRef;
  ControlRef controlRef;
  Str255     itemName;

  *pmDialog = gPMDialog;

  osStatus = PMGetDialogPtr(*pmDialog,&dialogRef);

  if(osStatus == kPMNoError)
  {
    // ..................................................................... append
DITL

    ditlHdl = GetResource('DITL',rPrintDialogAppendDITL);
    numberOfExistingItems = CountDITL(dialogRef);
    AppendDITL(dialogRef,ditlHdl,appendDITLBottom);
    gFirstAppendedItemNo = numberOfExistingItems + 1;
    // ........... create font menu and attach to popup button, set current font to first
item

    menuRef = NewMenu(mFont,NULL);
    CreateStandardFontMenu(menuRef,0,0,0,NULL);
    GetDialogItemAsControl(dialogRef,gFirstAppendedItemNo,&controlRef);
    SetControlMinimum(controlRef,1);
    numberOfMenuItems = CountMenuItems(menuRef);
    SetControlMaximum(controlRef,numberOfMenuItems);
    SetControlData(controlRef,kControlEntireControl,kControlPopupButtonMenuRefTag,
                   sizeof(menuRef),&menuRef);
    GetMenuItemText(menuRef,1,itemName);
    GetFNum(itemName,&gFontNumber);

    // ......................... set second radio button to on state and set current font
size

    GetDialogItemAsControl(dialogRef,gFirstAppendedItemNo + 2,&controlRef);
    SetControlValue(controlRef,1);
    gFontSize = 12;

    // ....................................................... switch fractional widths
off

    GetDialogItemAsControl(dialogRef,gFirstAppendedItemNo + 4,&controlRef);
    SetControlValue(controlRef,0);
    SetFractEnable(false);
  }

  if(osStatus == kPMNoError)
    osStatus = PMGetItemProc(*pmDialog,&gOldItemEvaluateFunctionUPP);
  if(osStatus == kPMNoError)
    osStatus = PMSetItemProc(*pmDialog,gNewItemEvaluateFunctionUPP);

  if(osStatus == kPMNoError)
    PMSetModalFilterProc(*pmDialog,gEventFilterUPP);

  if(osStatus != kPMNoError)
    doErrorAlert(osStatus);
}

// ****************************************************************
itemEvaluationFunction
```

```
void  itemEvaluationFunction(DialogRef dialogRef,SInt16 itemHit)
{
  SInt16     localizedItemNo, controlValue;
  ControlRef controlRef;
  MenuRef    menuRef;
  Str255     itemName;

  localizedItemNo = itemHit - gFirstAppendedItemNo + 1;

  if(localizedItemNo > 0)
  {
    if(localizedItemNo == iPopupButton)
    {
      GetDialogItemAsControl(dialogRef,gFirstAppendedItemNo,&controlRef);
      controlValue = GetControlValue(controlRef);
      GetControlData(controlRef,kControlEntireControl,kControlPopupButtonMenuHandleTag,
                     sizeof(menuRef),(Ptr) &menuRef,NULL);
      GetMenuItemText(menuRef,controlValue,itemName);
      GetFNum(itemName,&gFontNumber);
    }
    else if(localizedItemNo >= iRadioButton10pt && localizedItemNo <= iRadioButton14pt)
    {
      GetDialogItemAsControl(dialogRef,gFirstAppendedItemNo + 1,&controlRef);
      SetControlValue(controlRef,0);
      GetDialogItemAsControl(dialogRef,gFirstAppendedItemNo + 2,&controlRef);
      SetControlValue(controlRef,0);
      GetDialogItemAsControl(dialogRef,gFirstAppendedItemNo + 3,&controlRef);
      SetControlValue(controlRef,0);

      GetDialogItemAsControl(dialogRef,itemHit,&controlRef);
      SetControlValue(controlRef,1);

      if(localizedItemNo == iRadioButton10pt)
        gFontSize = 10;
      else if(localizedItemNo == iRadioButton12pt)
        gFontSize = 12;
      else if(localizedItemNo == iRadioButton14pt)
        gFontSize = 14;
    }
    else if(localizedItemNo == iCheckboxFracWidths)
    {
      GetDialogItemAsControl(dialogRef,gFirstAppendedItemNo + 4,&controlRef);
      SetControlValue(controlRef,!GetControlValue(controlRef));
      SetFractEnable(GetControlValue(controlRef));
    }
  }
  else
  {
    InvokePMItemUPP(dialogRef,itemHit,gOldItemEvaluateFunctionUPP);
  }
}

// ****************************************************************************
eventFilter

Boolean  eventFilter(DialogRef dialogRef,EventRecord *eventStrucPtr,SInt16 *itemHit)
{
  Boolean handledEvent;
  GrafPtr oldPort;

  handledEvent = false;

  if((eventStrucPtr->what == updateEvt) &&
     ((WindowRef) eventStrucPtr->message != GetDialogWindow(dialogRef)))
  {
    doUpdate(eventStrucPtr);
```

```
  }
  else
  {
    GetPort(&oldPort);
    SetPortDialogPort(dialogRef);

    handledEvent = StdFilterProc(dialogRef,eventStrucPtr,itemHit);

    SetPort(oldPort);
  }

  return handledEvent;
}

//
*****************************************************************************************
```

Demonstration Program CarbonPrinting Comments

When the program is run, the user should

- Choose Page Setup… from the File menu, make changes in the Page Setup dialog, and observe the information, extracted from the `PMPageFormat` object, drawn in the window titled Some Printing Information.

- Choose Print… from the File menu, note the items added to the Print dialog, select the desired font, font size, and fractional widths setting using these items, and print the document.

The user should print the document several times using different page size, scaling, and orientation settings in the Page Setup dialog (noting the changed information in the Printing Settings window), and occasionally limiting the number of changes printed by changing the start-page and end-page settings in the Print dialog.

The window in which the "document" is displayed is titled Simulated Document because, in this demonstration, the document is not loaded from a file. Rather, a document is simulated using text from a `'TEXT'` resource and a picture from a `'PICT'` resource. This approach is used in order to keep that part of the source code not related to printing per se to a minimum. For the same reason, the (flattened) `PMPageFormat` object is not saved to the resource fork of a document file in this demonstration. (The demonstration program in Chapter 19 shows how to do this.)

Printing.h

defines

Constants are established for the resource IDs of a `'PICT'` and `'TEXT'` resources used for the printing demonstration, and for the `'DITL'` resource containing the items to be appended to the Print dialog. Five constants are established for the item numbers of the items in the `'DITL'` resource. `kMargin` is used to set the margins for the printed page.

typedefs

A handle to a structure of type `docStructure` will be stored in the Window object for the Simulated Document window. The fields of this structure will be assigned `PMPrintSession`, `PMPageFormat`, and `PMPrintSettings` objects, together with handles to the `'TEXT'` and `'PICT'` resources referred to above.

CarbonPrinting.c

Global Variables

`gFontNumber` will be assigned the current font number. `gFontSize` will be assigned the current font size. Both of these values can be changed by the user via the items added to the Print dialog.

`gNewItemEvaluateFunctionUPP` and `gEventFilterUPP` will be assigned a universal procedure pointer to, respectively, the item evaluation function and event filter function for the customized Print dialog. `gPMDialog` will be assigned a pointer to a `PMDialog` object for the customized Print dialog.

The last four variables are incidental to the demonstration, and will be used to store information to be displayed in the second window (titled Some Printing Information) created by the program.

main

After the Simulated Document window is created, its font is set to Geneva 10 point and the global variables `gFontNumber` and `gFontSize` are set to reflect this. A document structure is then created and associated with the window. The three fields of the document structure which will be assigned Carbon Printing Manager objects are then initialized.

The Some Printing Information window is then created.

The call to `doGetDocument` displays the simulated document in the Simulated Document window.

Note that, in this program, error handling of all errors other than Carbon Printing Manager errors is somewhat rudimentary. The program simply exits.

doGetDocument

`doGetDocument` creates a monostyled `TextEdit` structure, assigns the handle to this structure to the relevant field of the Simulated Document window's document structure, loads a `'TEXT'` resource, and inserts it into a `TextEdit` structure. (The act of insertion causes the text to be drawn on the screen.) A `'PICT'` resource is then loaded and its handle assigned to the relevant field of the Simulated Document window's document structure. The call to `InvalWindowRect` invalidates the content region, generating an update event which, as will be seen, causes the picture to be drawn in the window.

`TextEdit` is not addressed until Chapter 21; however, to facilitate an understanding of

the TextEdit aspects of this program, it is sufficient at this stage to understand that a monostyled TextEdit structure contains the following fields:

`DestRect` The destination rectangle into which text is drawn. The bottom of the destination rectangle can extend to accommodate the end of the text. In other words, you can think of the destination rectangle as bottomless.

`ViewRect` The rectangle within which text is actually displayed.

`HText` A handle to the actual text.

`TxFont` The font number of the font to be used,

`txSize` The size of the font in points.

`LineHeight` The vertical spacing, in pixels, of the lines of text.

`NLines` The total number of lines of text.

`LineStarts` An array with a number of elements corresponding to the number of lines of text. Each element contains the offset of the first character in each line.

Note that the destination and view rectangles were passed in the call to `TENew`, these rectangles having been defined in the preceding five lines of code.

doUpdateDocumentWindow

As will be seen, two of the functions directly related to printing operations will change the values in the `TextEdit` structure's `destRect`, `txFont`, `txSize`, and `lineHeight` fields. The middle block of the `doUpdateDocumentWindow` code simply resets the values in these fields back to those that existed after `doGetDocument` was called, and then calls `TECalText` to re-calculate the line starts and `TEUpdate` to draw the text.

The second blocks saves the values in the `TextEdit` structure's `destRect`, `txFont`, `txSize`, and `lineHeight` fields as they existed before the update event was received and the fourth block restores these settings. The last two lines redraw the picture in the window.

doMenuChoice

If the user chose Page Setup… from the File menu, `doPageSetupDialog` is called to display the Page Setup dialog, handle user interaction within the dialog, and record the settings made by the user. If an error is returned, `doErrorAlert` is called to present an

alert advising of the error number.

If the user chose Print… from the File menu, `doPrintSettingsDialog` is to display the Print dialog, handle user interaction within the dialog, and record the settings made by the user. If no error is returned, `doPrinting` is then called to perform the printing. If `doPrinting` returns an error, `doErrorAlert` is called to present an alert advising of the error number.

If `doPrintSettingsDialog` returns an error, `doPrinting` is not called and `doErrorAlert` is called to present an alert advising of the error number.

doCreateOrValidatePageFormat

`doCreateOrValidatePageFormat` is the first of the major printing functions. It is called, as their first action, by `doPageSetUpDialog` and `doPrintSettingsDialog`-that is, whenever the user chooses Page Setup… or Print… from the File menu. The main purpose of this function is to create a `PMPageFormat` object, assign default parameter values to that object, and associate the object with the document's window or, if the object has previously been created, simply validate the existing object.

The call to `PMCreateSession` creates a printing session object.

If a `PMPageFormat` object is not currently assigned to the pageFormat field of the window's document structure, `PMCreatePageFormat` and `PMSessionDefaultPageFormat` are called to create a `PMPageFormat` object and assign default parameters to it. The object is then associated with the document's window by assigning the object to the relevant field of the window's document structure.

On the other hand, if a `PMPageFormat` object is currently assigned to the `pageFormat` field of the window's document structure, `PMSessionValidatePageFormat` is called to validate the object within the context of the current printing session.

If no errors occurred, the `PMPrintSession` object is assigned to the relevant field of the document window's document structure, otherwise both the `PMPrintSession` and `PMPageFormat` objects are released.

doPageSetUpDialog

`doPageSetUpDialog` is called when the user chooses Page Setup… from the File menu. Its purpose is to present the Page Setup dialog and modify the page setup information stored in the `PMPageFormat` object associated with the document's window according to settings made by the user within the dialog.

The call to `doCreateOrValidatePageFormat` ensures that `PMPageFormat` and `PMPrintSession` objects have been created and assigned to the relevant field of the window's document structure. The call to `PMSessionPageSetupDialog` then presents the Page Setup dialog and handles all user interaction within the dialog. If the user clicks the OK push button to dismiss the dialog, the `PMPageFormat` object will be updated to reflect any changed settings made by the user. No updating occurs if the user clicks the Cancel push button to dismiss the dialog.

When the dialog has been dismissed, `PMRelease` is called to release the `PMPrintSession` object.

doPrintSettingsDialog

`doPrintSettingsDialog` is called when the user chooses Print… from the File menu. Its main purpose is to create a `PMPrintSettings` object and associate it with the document's window, and present the Print dialog and modify the print settings information stored in the `PMPrintSettings` object according to changed settings made by the user within the dialog.

The call to `doCreateOrValidatePageFormat` ensures that `PMPageFormat` and `PMPrintSession` objects have been created and assigned to the relevant field of the window's document structure. The call to `PMCreatePrintSettings` then creates a `PMPrintSettings` object.

The call to `PMSetJobNameCFString` sets the print job name used by the printing system. `PMSessionDefaultPrintSettings` is then called to assign default parameter values (number of copies, page range, and so on) to the `PMPrintSettings` object, following which the PMPrintSettings object is assigned to the relevant field of the document window's document structure.

The next block executes only if the program is running on Mac OS X. `PMSetPageRange` sets the valid range of pages that can be printed to 1-32000. (If the user enters values outside this range in the Print dialog, the Carbon Printing Manager displays an "out-of-range" alert.) The next two calls set the default first and last page numbers to be drawn in the relevant edit text fields in the Print dialog. Note that, if `kPMPrintAllPages` is passed in `PMSetLastPage`'s last parameter, the All radio button in the Print dialog will be selected when the dialog is displayed.

The Print dialog to be presented is to be customized. Accordingly, the first three lines inside the next if block create universal procedure pointers for the initialization, item evaluation, and event filter functions contained in the source code file PrintDialogAppend.c.

`PMSessionPrintDialogInit` is called to initialize a custom Print dialog. On return, the global variable `gPMDialog` contains a pointer to an initialized `PMDialog` object,

ready for customization. This pointer is assigned to a global variable because the `PMSessionPrintDialogMain` function does not include a parameter for passing this `PMDialog` object to the dialog initialization function.

`PMSessionPrintDialogMain` presents the customized Print dialog, the universal procedure pointer to the initialization function being passed in the fifth parameter.

If the user clicked the Print push button, when `PMSessionPrintDialogMain` returns, the `PMPrintSettings` object will contain information on the settings displayed in the Print dialog (except for the settings in the customized section of the dialog).

If an error occurred, or if the user clicked the Cancel push button in the Print dialog, the PMPrintSettings and PMPrintSession objects are released and disassociated from the window. In addition, within the function doMenuChoice, the call to the function doPrinting will not occur.

doPrinting

`doPrinting` contains the printing loop.

After the current graphics port is saved for later restoration, `PMGetFirstPage` and `PMGetLastPage` are called to get the first and last page to print, as set in the Print dialog.

The next block is incidental to the demonstration, simply storing the first page, last page, and number of copies information retrieved from the `PMPrintSettings` object in global variables so that they can later be drawn in the Some Printing Information window.

`PMGetAdjustedPageRect` is then called to get the page rectangle, taking account of page orientation setting, scale setting, and drawing resolution. The double values in the fields of this `PMRect` structure are assigned to the (`SInt16`) fields of a normal `Rect` structure before that rectangle is passed to a function which calculates the actual number of pages in the document.

If the calculated actual number of pages is less than the value returned by the call to `PMGetLastPage`, the value in the variable `lastPage` is changed to the calculated number of pages.

For Mac OS X only, `PMSetFirstPage` and `PMSetLastPage` are called to set the first and actual last page in the `PMPrintSettings` object. This ensures that the page number information displayed in the progress dialog invoked during printing will be correct.

The printing loop is now entered. The first action is to call `PMSessionBeginDocument` to establish a graphics context (`PMPrintContext` object) for imaging the document. As will be seen, a function exists to obtain the graphics

port (that is, the printing port) associated with this object.

The while loop spools all of the pages in the document, relying on the Carbon Printing Manager to print the correct page range. Within the loop

- `GetPort` is called to save the current graphics port.
- `PMSessionBeginPage` is called to initialize the printing port. (Note that, for Mac OS 8/9, this function may also be used to initialize a scaling rectangle for drawing the page, in which case the address of a `PMRect` passed in the last parameter will be passed to the printer driver.)
- `PMSessionGetGraphicsContext` is called to obtain the current printing port, and `SetPort` is called to set this port as the current port.
- `doDrawPage` is called to draw the specified page in the current printing port.
- `SetPort` is called to restore the saved graphics port.
- `PMSessionEndPage` is called to print the page.

After the pages have been spooled, `PMSessionEndDocument` is called to close the printing port. On Mac OS X, this call also dismisses the progress dialog.

Note that the printing loop does not have to concern itself with the number of copies, since this is handled automatically by the Carbon Printing Manager.

Finally, the `PMPrintSettings` and `PMPrintSession` objects are released and disassociated from the document's window, and the graphics port saved at function entry is restored.

doCalcNumberOfPages

`doCalcNumberOfPages` is called by `doPrinting` to calculate the actual number of pages in the document based on the page rectangle passed to it.

The first line retrieves the handle to the specified window's document structure. The next two lines set the current font and font size to the font number and size set by the user in the appended items in the Print dialog. This allows the call to `GetFontInfo` to retrieve some relevant information about the font.

The next four lines change the values in the `txFont`, `txSize`, and `lineHeight` fields of the `TextEdit` structure whose handle is stored in the window's document structure. (Note that information obtained by the `GetFontInfo` call is used to calculate line height.)

The next three lines change the rectangle stored in the `destRect` field of the `TextEdit` structure to one equal to the received page rectangle less 180 pixels in width and 270 pixels in height. (This smaller rectangle is centered on the page rectangle both laterally and vertically.)

With these changes made, `TECalText` is called to recalculate the line starts. In addition to changing the values in the `lineStarts` array in the TextEdit structure, this call will assign the new total number of lines to the `nLines` field.

The matter of the actual calculation of the number of pages now follows. The first line in the last block gets the height of the previously defined destination rectangle. The next line calculates how many lines of text will fit into that height. The third line then calculates the total number of rectangles (and thus the number of pages) required to accommodate the whole of the text.

doDrawPage

`doDrawPage` is called by `doPrinting` to draw a specified page in the printing graphics port.

The first action is to set the printing graphics port's font and font size to the font number and size set by the user in the appended items in the Print dialog.

The next block retrieves a handle to the specified window's document structure, allowing handles to the `TextEdit` structure and picture to be retrieved.

In the next block, the destination rectangle in the `destRect` field of the `TextEdit` structure is assigned to a local variable, the height of this rectangle is assigned to a local variable, the lines per page is calculated and assigned to a local variable, and the total number of lines is assigned to a local variable.

In the next block, the first line gets the starting offset, that is, the offset from the first character in the block of text to the first character in the first line of text for the specified page number. The next four lines get the ending offset, that is, the offset to the last character in the last line of text for the specified page.

The call to `TENew` creates a new monostyled `TextEdit` structure with the previously defined destination rectangle passed in both the destination and view rectangle parameters. The following line gets a handle to the actual text in the `TextEdit` structure. This handle is then locked preparatory to a call to `TEInsert`. Using the offsets previously calculated, `TEInsert` then inserts the text for the current page into the newly created `TextEdit` structure, an action which causes that text to be drawn in the printing graphics port. The text having been drawn, the `TextEdit` structure is then disposed of.

If this is the first page, the next block draws the previously loaded picture at the top left of the previously defined rectangle.

The last three lines draw the page number at the bottom left of the original page rectangle.

doDrawPrintInfo and doDrawRectStrings

`doDrawPrintInfo` is called when an update event is received for the Some Printing Information window. Carbon Printing Manager accessor functions are then used to obtain information from the `PMPageFormat` object associated with the document's window and draw that information in the window. In addition, information retrieved from the `PMPrintSettings`, and saved to global variables within the function `doPrinting`, is drawn in the window if the global variable `doDrawPrintSettingsStuff` is set to true.

doErrorAlert

`doErrorAlert` is called when the printing functions return an error other than the "user cancelled" error. An alert showing the error code is displayed.

PrintDialogAppend.c

initializationFunction

Recall that, in the function `doPrintSettingsDialog`, a universal procedure pointer to `initializationFunction` was passed in the `myInitProc` parameter of the `PMSessionPrintDialogMain` call. `PMSessionPrintDialogMain` thus calls this function.

Recall also that the pointer to the initialized `PMDialog` object was assigned to a global variable (`gPMDialog`) because the `PMSessionPrintDialogMain` function does not include a parameter for passing a `PMDialog` object to a dialog initialization function. At the first line, the pointer to the `PMDialog` object is copied to the initialization function's formal parameter `pmDialog`.

The call to `PMGetDialogPtr` gets a reference to the Print dialog dialog object.

The DITL to be appended to the dialog contains the following items:

- A pop-up menu for font selection
- Three radio buttons for font size selection
- A checkbox for selecting fractional widths on or off
- A primary group box (text title variant)

`GetResource` loads the specified `'DITL'` resource and gets a handle to it. `CountDITL` counts the current number of items in the Print dialog. `AppendDITL` then appends the new items to the dialog. For some printers on Mac OS 8/9, this causes the dialog to expand downwards to accommodate the added items. For others (for, example, the LaserWriter 8),

and on Mac OS X, the result of the AppendDITL call is that a pane is created for the items and the name of the application is inserted into the menu of a pop-up group box. When the item containing the application's name is chosen from the pop-up menu, the pane is displayed and the appended items are accessible.

The global variable gFirstAppendedItemNo is then assigned the item number in the new Print dialog item list of the first appended item (the pop-up menu button). This will be required by the function itemEvaluationFunction.

The next block calls CreateStandardFontMenu to create the menu for the pop-up menu button, which is then assigned to the pop-up menu button by SetControlData. Note that the font number for the first item in the menu is assigned to the global variable gFontNumber.

The next block selects the second radio (12pt) button and sets the global variable gFontSize to 12. The next block unchecks the checkbox and sets fractional widths to off.

The printer driver's item evaluation function will be called upon by the item evaluation function (itemEvaluationFunction) to handle mouse-downs in the Print dialog's standard items. Accordingly, PMGetItemProc is called to assign the universal procedure pointer to the driver's evaluation function to a global variable for later use. The call to PMSetItemProc makes itemEvaluationFunction the current item evaluation function.

Finally, the call to PMSetModalFilterProc makes the application-defined (callback) function eventFilter the event filter function for the Print dialog.

itemEvaluationFunction

itemEvaluationFunction handles item hits in the Print dialog. The item number of the item hit is received in the second parameter.

At the first line, the item number of the item hit is "localized". This means that, for example, the localized item number of the pop-up menu button will be 1. In other words, if the localized item number is greater than 0, it will be the item number of one of the appended items; otherwise, it will be the item number of one of the Print dialog's standard items.

If the localized item number is greater than 0, and if it is the localized item number for the pop-up menu button, the control's value (that is, the menu item number) is retrieved. GetMenuItemText is called to get the text of the menu item, and GetFNum is called to get the font number for this font and assign it to the relevant global variable.

If the localized item number is the localized item number for one of the radio buttons, all

radio buttons are unchecked, the radio button hit is checked, and the global variable which holds the current text size is assigned the appropriate value. If the localized item number is the localized item number for the checkbox, the current value of that control is flipped and `SetFractEnable` is called to set fractional widths on or off as appropriate.

If the localized item number is 0 or less, the item must be one of the Print dialog's standard items. In this case, the printer driver's item evaluation function is called upon to handle the item hit.

eventFilter

`eventFilter` is identical to the custom event filter for modal dialogs introduced in Chapter 8. The use of a the event filter is optional. Its use in this program simply allows the Print dialog, together with windows belonging to background applications, to receive update events (required only on Mac OS 8/9).

The Page Setup and Print dialogs in this demonstration program are application-modal. This is because the `AppendDITL` method used to customize the Print dialog prevents that dialog from being created as a window-modal (sheet) dialog. The basic modifications required to cause the dialogs to be window-modal are as follows:

- Display the Print dialog using the function `PMSessionPrintDialog` and eliminate all code relating to dialog customization.
- Immediately before the calls to `PMSessionPageSetupDialog` and `PMSessionPrintDialog`, call the function `PMSessionUseSheets`, passing the parent window's reference in the `documentWindow` parameter and a universal procedure pointer to an application-defined (callback) function in the `sheetDoneProc` parameter.
- Add two application-defined (callback) functions that perform the actions required immediately following the dismissal of the dialogs, and modify the existing post-dismissal code accordingly.

Chapter 16. More on Windows

IN THIS CHAPTER

- Introduction
- Floating Windows
- Utility and Toolbar Windows
- Window Proxy Icons
- Window Path Pop-Up Menus
- Transitional Window Animation and Sounds
- Creating and Storing Windows
- Accessing Window Information
- Moving and Positioning Windows
- Associating Data with Windows
- Adding To and Removing From the Update Region
- Setting Content Region Color and Pattern
- Window Scrolling
- The Window Menu
- Live Window Resizing
- Main Constants, Data Types, and Functions
- Demonstration Program Windows2 Listing
- Demonstration Program Windows2 Comments

Introduction

As stated in Chapter 4, Mac OS 8.5 introduced a number of new features and associated system software functions to the Window Manager. The new functions associated with zooming and re-sizing windows were addressed in Chapter 4. This chapter addresses the remaining additional features, which include:

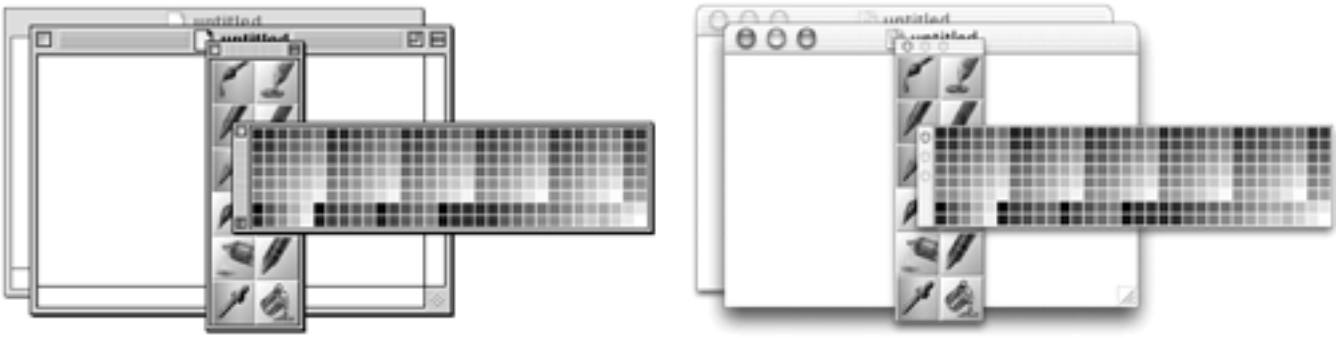
- Support for
    - Floating windows.
    - Window proxy icons.
    - Window path pop-up menus.
    - Transitional window animations and sounds.
- New functions for
    - Creating and storing windows.
    - Accessing window information.
    - Moving and positioning windows.
    - Associating data with a window.
    - Adding and removing rectangles and regions to and from a window's update region.
    - Setting the color or pattern of a window's content region.

This chapter also addresses additional features introduced with Carbon and live window resizing introduced with Mac OS X.

Floating Windows

Floating windows are windows that stay in front of all of an application's document windows. They are typically used to display tool, pattern, color, and other choices to be made available to the user. Examples of floating windows are shown in Fig 1.

Figure 1. FLOATING WINDOWS



In terms of front-to-back ordering of onscreen objects, floating windows, unlike document windows, are all basically equal. Unless they actually overlap each other, there is no visual cue of any front-to-back ordering as there is with normal windows (see Fig 1). Because of this equality, floating windows almost always appear in the active state. The exception is when a modal or movable modal dialog or alert is presented to the user. When this occurs, the appearance of all floating windows changes to reflect the inactive state.

Floating Window Types

The 16 available window types for floating windows are shown in Figs 4 and 5 in Chapter 4.

Opening, Closing, Showing, and Hiding Floating Windows

Floating windows may be created using the function `CreateNewWindow` (see below) with the constant `kFloatingWindowClass` passed in the `windowClass` parameter.

When a floating window is created, it should remain open until the application is closed, and your application should provide the user with a means to hide or show the window as and when required. Ordinarily, it should do this by providing an item in an appropriate menu that allows the user to toggle the window between the hidden and showing states.

A floating window's close box/button should simply hide the window, not close it. For that reason, the close box/button in floating windows should be conceived of as a "hide" box/button

rather than as a close box/button.

Floating windows should be hidden when the application receives a suspend event. This is to avoid user confusion arising from one application's floating windows being visible when another application is in the foreground. The application's floating windows should be shown again only when the application receives a subsequent resume event.

Carbon Note

In Carbon applications, floating windows are hidden and shown automatically on suspend and resume events. It is thus not necessary for Carbon applications to call `HideFloatingWindows` and `ShowFloatingWindows`.

Functions Relating to Floating Windows

The following function is relevant to floating windows:

Function	*Description*
`AreFloatingWindowsVisible`	Indicates whether an application's floating windows are visible.

Utility and Toolbar Windows

Carbon introduced the **utility window** (a system-wide floating window which floats above all other windows) and the **toolbar window,** which floats above all document windows in an application but below floating windows. (See Window Class Constants, later in this chapter.)

Window Proxy Icons

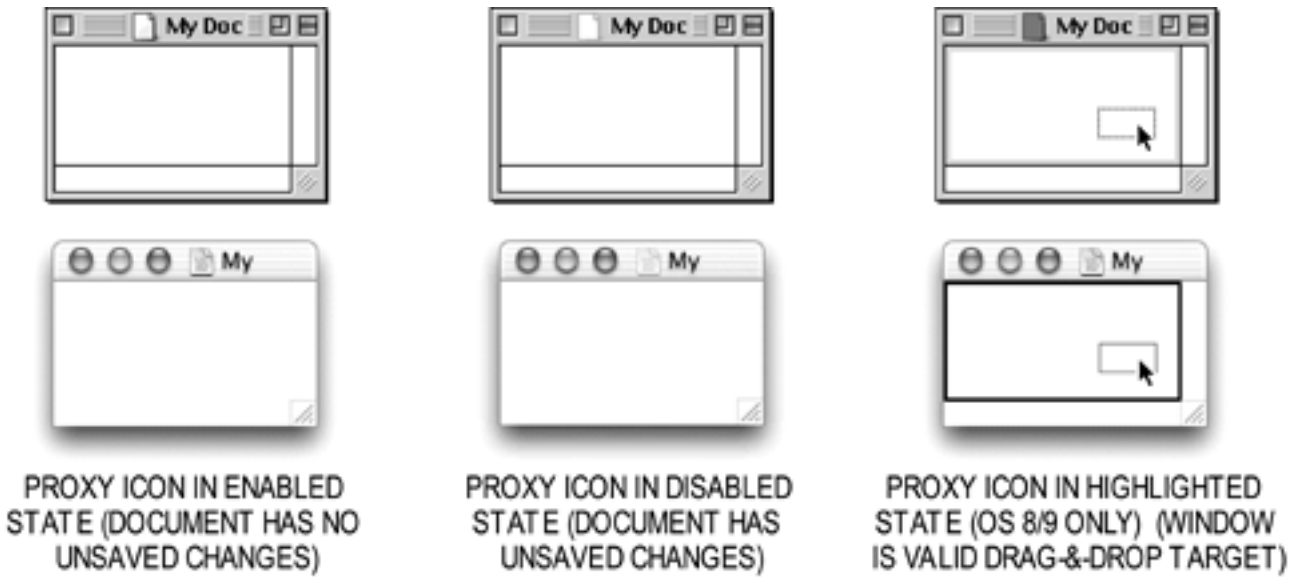
Window proxy icons are small icons displayed in the title bar of document windows. Ordinarily, a specific document file is associated with a specific window, and the proxy icon serves as a proxy for the document file's icon in the Finder.

Proxy icons

- May be dragged, in the same way that the document's icon in the Finder may be dragged, so as to move or copy the document file.
- Provide visual feedback to the user on the current state of the document. For example, when the document has unsaved changes, your application should cause the proxy icon to be displayed in the disabled state, thus preventing the user from dragging it. (Unsaved documents should not be capable of being moved or copied.)
- Provide visual feedback to the user indicating that the document window is a valid drag-and-drop target. In this case, your application should cause the proxy icon to appear in the highlighted state.

Fig 2 shows a typical window proxy icon for a document in the enabled, disabled, and highlighted states.

Figure 2. WINDOW PROXY ICONS


My Doc
My
PROXY ICON IN ENABLED STATE (DOCUMENT HAS NO UNSAVED CHANGES)
PROXY ICON IN DISABLED STATE (DOCUMENT HAS UNSAVED CHANGES)
PROXY ICON IN HIGHLIGHTED STATE (OS 8/9 ONLY) (WINDOW IS VALID DRAG-&-DROP TARGET)

In Fig 2, note that in the drag and drop operation depicted at the right, the window's content area is highlighted along with the proxy icon. Applications typically call the Drag Manager function `ShowDragHilite` to indicate, with this highlighting, that a window is a valid

drag-and-drop target. `ShowDragHilite` and `HideDragHilite` highlight and unhighlight the proxy icon as well as the content area.

Changing the State of a Proxy Icon

Applications typically keep track of the modification state of a document so as to inform users that they has made changes to the document which they might wish to save before closing the document's window, for example. When a document has unsaved changes, your application should call `SetWindowModified` with `true` passed in the `modified` parameter to cause the proxy icon to appear in the disabled state. (On Mac OS X, this is accompanied by a dot appearing in the middle of the close button.) When the changes have been saved, your application should call `SetWindowModified` with `false` passed in the `modified` parameter to cause the proxy icon to appear in the enabled state.

Handling Mouse-Down Events in a Window Proxy Icon

When a mouse-down event occurs in your application's window, and when `FindWindow` returns the `inProxyIcon` result code, your application should simply call `TrackWindowProxyDrag`. `TrackWindowProxyDrag` handles all aspects of the drag process while the user drags the proxy icon.

Proxy Icons and File Synchronization Functions

It is always possible that, while a document file is open, the user may drag its Finder icon to another folder (including the Trash) or change the name of the file via the Finder icon. The application itself has no way of knowing that this has happened and will assume, unless it is informed otherwise, that the document's file is still at its original location with its original name. For this reason applications often include a frequently called **file synchronization function,** which synchronizes the application with the actual current location (and name) of its currently open document files.

A document's proxy icon is much more prominent to the user than the document's Finder icon. Thus, when proxy icons are used, there is an even greater possibility that the user will move the file represented by the proxy icon to a different folder while the document is open. The provision of a file synchronization function is therefore imperative when proxy icons are implemented.

File synchronization functions are addressed in Chapter 18.

Functions Relating to Window Proxy Icons

The following functions are relevant to window proxy icons:

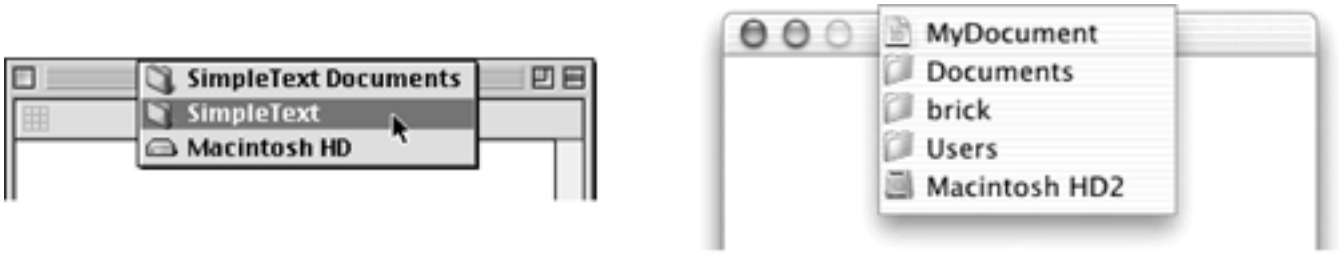
Function	*Description*
`SetWindowProxyCreatorAndType`	Sets the proxy icon for a window that has no associated file. New untitled windows should have a proxy icon so as to be consistent, in terms of appearance, with other windows.
`SetWindowProxyFSSpec`	Associates a file with a window using a file system specification (`FSSpec`) structure, thus establishing a proxy icon for the window.
`GetWindowProxyFSSpec`	Gets a file system specification (`FSSpec`) structure for the file associated with a window.
`SetWindowProxyAlias`	Associates a file with a window using a handle to an `AliasRecord` structure, thus establishing a proxy icon for the window.
`GetWindowProxyAlias`	Gets alias data for the file associated with a window.
`SetWindowProxyIcon`	Overrides the default proxy icon for a window.
`GetWindowProxyIcon`	Gets a window's proxy icon.
`RemoveWindowProxy`	Dissociates a file from a window.
`TrackWindowProxyDrag`	Handles all aspects of the drag process when a proxy icon is dragged by the user.

Note that `SetPortWindowPort` (or `SetPort`) should be called to set the relevant window's graphics port as the current port before calling `SetWindowProxyCreatorAndType`, `SetWindowProxyFSSpec`, `SetWindowProxyAlias`, and `SetWindowProxyIcon`.

Window Path Pop-Up Menus

If your application supports window path pop-up menus, when the user presses the Command key and clicks a window's title, your window displays a pop-up menu containing a standard file system path. The pop-up menu allows the user to open windows for folders along the file system path. An example of a window path pop-up menu is shown in Fig 3.

Figure 3. WINDOW PATH POP-UP MENUS



Displaying and Handling a Window Path Pop-Up Menu

The proxy icon region overlays the title text region which, in turn, overlays the drag region (see Fig 4). Your application must be prepared to respond to a Command-click in either region.

Figure 4. DRAG, TITLE TEXT, AND PROXY ICON REGIONS



When `FindWindow` returns the `inProxyIcon` part code, and `TrackWindowProxyDrag` returns `errUserWantsToDragWindow`, your application should proceed on the assumption that the `inDrag` part code was returned by `FindWindow`.

When `FindWindow` returns the `inDrag` part code, your application should call `IsWindowPathSelectClick` to determine whether the mouse-down event should activate the window path pop-up menu. If `IsWindowPathSelectClick` returns `true`, `WindowPathSelect` should be called to display the menu.

If the user chooses a menu item for a folder, your application must ensure that the associated window is visible by calling a function that makes the Finder the frontmost process.

Window path pop-up menus are demonstrated at the demonstration program associated with Chapter 18.

Transitional Window Animation and Sounds

On Mac OS 8/9, prior to Mac OS 8.5, the Window Manager supported the playing of a sound to accompany the animation that occured when a user clicked a window's collapse box. Mac OS 8.5 added support for animation and sounds to accompany the hiding and showing of windows.

The function `TransitionWindow` may be used in lieu of the older functions `HideWindow` and `ShowWindow` to hide and show windows. `TransitionWindow` causes a transitional animation to be displayed, a transitional sound to be played (on Mac OS 8/9), and the necessary update and activate events to be generated.

Creating and Storing Windows

Mac OS 8.5 introduced the following functions for creating and storing windows:

Function	*Description*
`CreateNewWindow`	Creates a window from parameter data.
`CreateWindowFromResource`	Creates a window from `'wind'` resource data.
`CreateWindowFromCollection`	Creates a window from collection data.
`StoreWindowIntoCollection`	Stores data describing a window into a collection.

Use of the last three of these functions requires a basic understanding of **collections, flattened collections** and **`'wind'` resources.**

Collections, Flattened Collections, and 'wind' Resources

Collections

A **collection object** (or, simply, a collection) is an abstract data type that allows you to store multiple pieces of related information.

A collection is like an array in that it contains a number of individually accessible items. However, unlike an array, a collection allows for a variable number of data items and variable-size items. A collection is also similar to a database, in that you can store information and retrieve it using a variety of search mechanisms.

The internal structure of a collection is private. This means that you must store information in a collection and retrieve information from it using Collection Manager functions.

Your application can store a window in a collection using the function `StoreWindowIntoCollection`. (This applies to any window, not just windows created using the new functions introduced with Mac OS 8.5.) Data associated with the window (for example, text) may also be stored in the same collection.

Using the function `CreateWindowFromCollection`, you can create a window from collection data. Note that `CreateWindowFromCollection` creates the window invisibly. After creating the window, you must call the function `TransitionWindow` to display the window.

Flattened Collections

Using the Collection Manager, your application can create a flattened collection from a collection. A flattened collection is a stream of address-independent data.

The 'wind' Resource

The `'wind'` resource consists of an extensible flattened collection. Using the Resource Manager, your application can store a flattened collection, consisting of a window and its data, into a `'wind'` resource.

Using the function `CreateWindowFromResource`, you can create a window from a `'wind'` resource. Note that `CreateWindowFromResource` creates the window invisibly. After creating the window, you must call the function `TransitionWindow` to display the window.

The CreateNewWindow Function

The `CreateNewWindow` function creates a window based on the class and attributes you specify in the `windowClass` and `attributes` parameters. The following constants may be passed in these parameters.

Window Class Constants

Constant	*Value*	*Description*
`kAlertWindowClass`	`1L`	Alert window.
`kMovableAlertWindowClass`	`2L`	Movable alert window.
`kModalWindowClass`	`3L`	Modal dialog window.
`kMovableModalWindowClass`	`4L`	Movable modal dialog window.
`kFloatingWindowClass`	`5L`	Floating window.
`kDocumentWindowClass`	`6L`	Document window or modeless dialog window. Note: The Window Manager assigns this class to windows created using the older window creation functions.
`kUtilityWindowClass`	`8L`	System-wide floating window.
`kHelpWindowClass`	`10L`	Help window.
`kSheetWindowClass`	`11L`	Sheet window. (Mac OS X only.)
`kToolbarWindowClass`	`12L`	Toolbar windows (above documents, below floating windows).
`kPlainWindowClass`	`13L`	Plain window (in document layer).
`kOverlayWindowClass`	`14L`	Transparent window that allows "screen" drawing via CoreGraphics. (Mac OS X only.)
`kSheetAlertWindowClass`	`15L`	Sheet windows for alerts. (Mac OS X only.)
`kAltPlainWindowClass`	`16L`	Alternate plain window (in document layer).
`kAllWindowClasses`	`0xFFFFFFFF`	For use with `GetFrontWindowOfClass`, `FindWindowOfClass`, `GetNextWindowOfClass` (see below).

Window Attribute Constants

Constant	*Bit*	*Description*
`kWindowNoAttributes`	`0L`	No attributes.

`kWindowCloseBoxAttribute`	`1L << 0`	Has close box/button.
`kWindowHorizontalZoomAttribute`	`1L << 1`	Has horizontal zoom box.
`kWindowVerticalZoomAttribute`	`1L << 2`	Has vertical zoom box.
`kWindowFullZoomAttribute`	`kWindowVerticalZoomAttribute | kWindowHorizontalZoomAttribute`	Has full zoom box/ zoom button.
`kWindowCollapseBoxAttribute`	`1L << 3`	Has a collapse box/ minimize button.
`kWindowResizableAttribute`	`1L << 4`	Has size box/ resize control.
`kWindowSideTitlebarAttribute`	`1L << 5`	Has side title bar. This attribute may be applied only to floating windows.
`kWindowNoUpdatesAttribute`	`1L << 16`	Does not receive update events.
`kWindowNoActivatesAttribute`	`1L << 17`	Does not receive activate events.
`kWindowNoShadowAttribute`	`1L << 21`	No shadow. (Mac OS X only.)
`kWindowHideOnSuspendAttribute`	`1L << 24`	Window is automatically hidden and shown on, respectively, suspend and shown on resume. (Carbon only.)

`kWindowStandardHandlerAttribute`	`1L << 25`	Window should have standard window event handler installed. (Carbon only.)
`kWindowHideOnFullScreenAttribute`	`1L << 26`	Window is automatically hidden during fullscreen mode. (Carbon only.)
`kWindowInWindowMenuAttribute`	`1L << 27`	Window is automatically tracked in Window menu. (Document windows are automatically given this attribute.)
`kWindowLiveResizeAttribute`	`1L << 28`	Window supports live resizing. (Mac OS X only.)
`kWindowStandardDocumentAttributes`	`kWindowCloseBoxAttribute | kWindowFullZoomAttribute | kWindowCollapseBoxAttribute | kWindowResizableAttribute`	Has standard document window attributes, that is, close box, full zoom box, collapse box, and size box.
`kWindowStandardFloatingAttributes`	`kWindowCloseBoxAttribute | kWindowCollapseBoxAttribute`	Has standard floating window attributes, that is, close box and collapse box.

Note that `CreateNewWindow` creates the window invisibly. After creating the window, you must call the `TransitionWindow` function to display the window.

Accessing Window Information

The following functions are provided for accessing window information:

Function	*Description*
`GetWindowClass`	Obtains the class of a window.
`GetWindowAttributes`	Obtains the attributes of a window.
`ChangeWindowAttributes`	Change the attributes of a window.
`IsValidWindowPtr`	Reports whether a reference is a valid window reference.
`FrontNonFloatingWindow`	Returns a reference to the frontmost visible window that is not a floating window.
`FindWindowOfClass`	A version of the `FindWindow` function that limits the search to windows of one particular class. If a window is found at the specified point, but is not of the specified class, `errWindowNotFound` is returned and the value of `outWindow` is set to `NULL`.
`GetFrontWindowOfClass`	A more explicit version of the `FrontWindow` and `FrontNonFloatingWindow` functions.
`GetNextWindowOfClass`	A more explicit version of the function `GetNextWindow`.

Moving and Positioning Windows

Moving Windows

When your application wishes to move a window for a reason other than a user-instigated drag, it should use the function `MoveWindowStructure` or the earlier function `MoveWindow`. `MoveWindow` repositions a window's content region, whereas `MoveWindowStructure` repositions a window's structure region.

The `SetWindowBounds` function provides a means to set the size of a window in addition to simply repositioning it. The size and position of the window are specified in a rectangle passed in the `globalBounds` parameter. In addition, you may specify whether this rectangle represents the bounds of the content region or the bounds of the structure region by passing either `kWindowContentRgn` or `kWindowStructureRgn` in the `regionCode` parameter. The sister function `GetWindowBounds` obtains the size and position of the bounding rectangle of the specified window region.

Positioning Windows

Generally speaking, a new window should be placed on the desktop where the user expects it to appear. For new document windows, this usually means just below and to the right of the last document window in which the user was working, although this is not necessarily the case on systems with multiple monitors.

The `RepositionWindow` function allows you to position a window relative to another window or a display screen. The required window positioning method may be specified by passing one of the following constants in the `method` parameter.

Window Positioning Constants

Constant	*Value*	*Description*
`kWindowCenterOnMainScreen`	`0x00000001`	The window is centered on the screen that contains the menu bar.

`kWindowCenterOnParentWindow`	`0x00000002`	The window is centered on the parent window. If the window is wider than the parent window, its left edge is aligned with the parent window's left edge.
`kWindowCenterOnParentWindowScreen`	`0x00000003`	The window is centered on the screen containing the parent window.
`kWindowCascadeOnMainScreen`	`0x00000004`	The window is placed just below the menu bar at the left edge of the main screen. Subsequent windows are placed relative to the first window in such a way that the frame of the preceding window remains

		visible behind the current window.
`kWindowCascadeOnParentWindow`	`0x00000005`	The window is placed below and to the right of the upper-left corner of the parent window in such a way that the frame of the parent window remains visible behind the current window.
`kWIndowCascadeOnParentWindowScreen`	`0x00000006`	The window is placed just below the menu bar at the left edge of the screen that contains the parent window. Subsequent windows are placed on the screen relative to the first window in such a way that the

		frame of the preceding window remains visible behind the current window.
`kWindowAlertPositionOnMainScreen`	`0x00000007`	The window is centered horizontally and positioned vertically, on the screen that contains the menu bar in such a way that about one-fifth of the screen is above it.
`kWindowAlertPositionOnParentWindow`	`0x00000008`	The window is centered horizontally and positioned vertically in such a way that about one-fifth of the parent window is above it.

`kWindowAlertPositionOnParentWindowScreen`	`0x00000009`	The window is centered horizontally and positioned vertically in such a way that about one-fifth of the screen containing the parent window is above it.

These constants should not be confused with the older positioning specification constants (see Chapter 4), and should not be used where those older constants are required (for example, in `'WIND'`, `'DLOG'`, and `'ALRT'` resources, and in the `StandardAlert` function).

Associating Data with Windows

The function `SetWRefCon` has always allowed your application to associate a pointer to data with a reference to a window object. An alternative method of associating data with windows is to use the standard mechanism introduced with Mac OS 8.5. (Both methods, incidentally, are Carbon-compliant.)

The following functions are provided for associating data with windows:

Function	*Description*
`SetWindowProperty`	Associates data with a window.
`GetWindowProperty`	Gets data associated with a window.
`GetWindowPropertySize`	Gets the size of data associated with a window.
`RemoveWindowProperty`	Removes data associated with a window.

Adding To and Removing From the Update Region

The Mac OS 8.5 Window Manager provided enhanced functions for manipulating the update region. Unlike their pre-Mac OS 8.5 counterparts (`InvalRect`, `InvalRgn`, `ValidRect`, and `ValidRgn`, which are not included in the Carbon API), the new functions allow the window on which they operate to be explicitly specified, meaning that they do not require the graphics port to be set prior to their use.

The following are the functions for manipulating the update region:

Function	*Description*
`InvalWindowRect`	Adds a rectangle to the window's update region.
`InvalWindowRgn`	Adds a region to the window's update region.
`ValidWindowRect`	Removes a rectangle from the window's update region.
`ValidWindowRgn`	Removes a region from the window's update region.

Setting Content Region Color and Pattern

The following functions set the color or pattern of a window's content region:

Function	*Description*
`SetWindowContentColor`	Sets the color to which a window's content region is redrawn on receipt of an update event.
`GetWindowContentColor`	Gets the color to which a window's content region is redrawn.
`SetWindowContentPattern`	Sets the pattern to which a window's content region is redrawn on receipt of an update event.
`GetWindowContentPattern`	Gets the pattern to which a window's content region is redrawn.

These functions do not affect the graphics port's background color or pattern.

Window Scrolling

The following functions scroll pixels within a window:

Function	*Description*
`ScrollWindowRect` `ScrollWindowRegion`	Scrolls pixels that are inside the specified rectangle (`ScrollWindowRect`)or region (`ScrollWindowRegion`) The pixels are shifted a distance of `inHPixels` horizontally and `inVPixels` vertically. The positive directions are to the right and down. The pixels that are shifted out of the specified window are not displayed, and the bits they represent are not saved. The exposed empty area created by the scrolling may be added to the update region or erased to the background color/pattern of the window's graphics port.

The following constants may be passed in the `inOptions` parameter of these functions:

Constant	*Meaning*
`kScrollWindowNoOptions`	No options.
`kScrollWindowInvalidate`	Add the exposed area to the window's update region.
`kScrollWindowEraseToPortBackground`	Erase the exposed area using the background color/pattern of the window's graphics port.

The Window Menu

Carbon introduced the system-managed Window menu, which is created using the function `CreateStandardWindowMenu`. After creating the menu you should add it to the menu list using `InsertMenu`. Menu items containing the titles of your application's windows will be automatically added to, and deleted from, the menu when those windows are created and closed. Floating windows will not be added to the menu.

It is not necessary to set the `kWindowInWindowMenuAttribute` attribute on your document windows in order for them to be added to the menu. Document windows are automatically given this attribute.

Using `SetWindowAlternateTitle` you can override the title displayed in the Window menu. You would ordinarily do this if the window title was not expressive enough.

Customizing the Window Menu

You can insert your own items to the Window menu by searching for the item with command ID `'wldv'` and inserting your items before that item (that is, immediately before the individual window items). `'wldv'` is the command ID of the divider that separates the window commands from the individual window items.

A problem here is that, at the time of writing, the divider had the `'wldv'` command ID on Mac OS X but not in CarbonLib.

You can append your own items at the end of the Window menu by searching for the item with command ID `'wlst'` and appending your items after that item. `'wlst'` is the command ID of a hidden menu item that marks the end of the individual window items.

Live Window Resizing

On Mac OS X, windows on which the `kWindowLiveResizeAttribute` attribute is set support live resizing. When this attribute is set, the window is continually redrawn while it is being resized, as opposed to just outlines of the window, title bar, and resize control being drawn. Full implementation of live resizing requires that the contents of the content region also be continually redrawn as the window is being resized. This requires the use of the Carbon event model.

The demonstration program CarbonEvents2 (Chapter 17) demonstrates live resizing.

Main Constants, Data Types, and Functions

Constants

Window Class

```
KAlertWindowClass                       = 1L
kMovableAlertWindowClass                = 2L
kModalWindowClass                       = 3L
kMovableModalWindowClass                = 4L
kFloatingWindowClass                    = 5L
kDocumentWindowClass                    = 6L
kUtilityWindowClass                     = 8L
kHelpWindowClass                        = 10L
kSheetWindowClass                       = 11L
kToolbarWindowClass                     = 12L
kPlainWindowClass                       = 13L
kOverlayWindowClass                     = 14L
kSheetAlertWindowClass                  = 15L
kAltPlainWindowClass                    = 16L
kAllWindowClasses                       = 0xFFFFFFFF
```

Window Attributes

```
kWindowNoAttributes                     = 0L
kWindowCloseBoxAttribute                = 1L << 0
kWindowHorizontalZoomAttribute          = 1L << 1
kWindowVerticalZoomAttribute            = 1L << 2
kWindowFullZoomAttribute                = kWindowVerticalZoomAttribute |
                                          kWindowHorizontalZoomAttribute
kWindowCollapseBoxAttribute             = 1L << 3
kWindowResizableAttribute               = 1L << 4
kWindowSideTitlebarAttribute            = 1L << 5
kWindowNoUpdatesAttribute               = 1L << 16
kWindowNoActivatesAttribute             = 1L << 17
kWindowNoShadowAttribute                = 1L << 21
kWindowHideOnSuspendAttribute           = 1L << 24
kWindowStandardHandlerAttribute         = 1L << 25
kWindowHideOnFullScreenAttribute        = 1L << 26
kWindowInWindowMenuAttribute            = 1L << 27
kWindowLiveResizeAttribute              = 1L << 28
kWindowStandardDocumentAttributes       = kWindowCloseBoxAttribute |
                                          kWindowFullZoomAttribute |
                                          kWindowCollapseBoxAttribute |
                                          kWindowResizableAttribute
kWindowStandardFloatingAttributes       = kWindowCloseBoxAttribute |
                                          kWindowCollapseBoxAttribute
```

Window Positioning

```
kWindowCenterOnMainScreen               = 0x00000001
kWindowCenterOnParentWindow             = 0x00000002
kWindowCenterOnParentWindowScreen       = 0x00000003
kWindowCascadeOnMainScreen              = 0x00000004
kWindowCascadeOnParentWindow            = 0x00000005
kWIndowCascadeOnParentWindowScreen      = 0x00000006
kWindowAlertPositionOnMainScreen        = 0x00000007
kWindowAlertPositionOnParentWindow      = 0x00000008
kWindowAlertPositionOnParentWindowScreen = 0x00000009
```

Window Transition Action and Effect

```
kWindowShowTransitionAction             = 1
kWindowHideTransitionAction             = 2
kWindowZoomTransitionEffect             = 1
```

Window Scrolling

```
KScrollWindowNoOptions                  = 0
KScrollWindowInvalidate                 = (1L << 0)
KscrollWindowEraseToPortBackground      = (1L << 1)
```

Data Types

Property Types

```
typedef OSType PropertyCreator;
typedef OSType PropertyTag;
```

Window Class and Attributes

```
typedef UInt32 WindowClass;
typedef UInt32 WindowAttributes;
```

Window Positioning

```
typedef UInt32 WindowPositionMethod;
```

Window Transitioning

```
typedef UInt32 WindowTransitionEffect;
typedef UInt32 WindowTransitionAction;
```

Window Scrolling

```
typedef UInt32 ScrollWindowOptions;
```

Functions

Floating Windows

```
Boolean   AreFloatingWindowsVisible(void);
```

Window Proxy Icons

```
OSStatus  SetWindowProxyCreatorAndType(WindowRef window,OSType fileCreator,OSType fileType,
          SInt16 vRefNum);
OSStatus  SetWindowProxyFSSpec(WindowRef window,const FSSpec *inFile);
OSStatus  GetWindowProxyFSSpec(WindowRef window,FSSpec *outFile);
OSStatus  GetWindowProxyAlias(WindowRef window,AliasHandle *alias);
OSStatus  SetWindowProxyAlias(WindowRef window,AliasHandle alias);
OSStatus  SetWindowProxyIcon(WindowRef window,IconRef icon);
OSStatus  GetWindowProxyIcon(WindowRef window,IconRef *outIcon);
OSStatus  RemoveWindowProxy(WindowRef window);
OSStatus  TrackWindowProxyDrag(WindowRef window,Point startPt);
```

Window Path Pop-Up Menus

```
Boolean   IsWindowPathSelectClick(WindowRef window,EventRecord *event);
OSStatus  WindowPathSelect(WindowRef window,MenuHandle menu,SInt32 *outMenuResult);
```

Transitional Window Animations and Sounds

```
OSStatus  TransitionWindow(WindowRef window,WindowTransitionEffect effect,
          WindowTransitionAction action,const Rect *rect);
```

Creating and Storing Windows

```
OSStatus  CreateNewWindow(WindowClass windowClass,WindowAttributes attributes,
          const Rect *bounds,WindowRef *outWindow);
OSStatus  CreateWindowFromResource(SInt16 resID,WindowRef *outWindow);
OSStatus  CreateWindowFromCollection(Collectioncollection,WindowRef *outWindow);
OSStatus  StoreWindowIntoCollection(WindowRef window,Collection collection);
```

Accessing Window Information

```
OSStatus  GetWindowClass(WindowRef window,WindowClass *outClass);
OSStatus  GetWindowAttributes(WindowRef window,WindowAttributes *outAttributes);
OSStatus  ChangeWindowAttributes(WindowRef window,WindowAttributes setTheseAttributes,
          WindowAttributes clearTheseAttributes);
Boolean   IsValidWindowRef(GrafPtr grafPort);
WindowRef FrontNonFloatingWindow(void);
OSStatus  FindWindowOfClass(const Point *where,WindowClass inWindowClass,
          WindowRef *outWindow,WindowPartCode *outWindowPart)
WindowRef GetFrontWindowOfClass(WindowClass inWindowClass,Boolean mustBeVisible);
WindowRef GetNextWindowOfClass(WindowRef inWindow,WindowClass inWindowClass,
          Boolean mustBeVisible);
```

Moving and Positioning Windows

```
OSStatus  MoveWindowStructure(WindowRef window,short hGlobal,short vGlobal);
OSStatus  SetWindowBounds(WindowRef window,WindowRegionCode regionCode,
          const Rect *globalBounds);
OSStatus  GetWindowBounds(WindowRef window,WindowRegionCode regionCode,Rect *globalBounds);
OSStatus  RepositionWindow(WindowRef window,WindowRef parentWindow,
          WindowPositionMethod method);
```

Associating Data with Windows

```
OSStatus  SetWindowProperty(WindowRef window,PropertyCreator propertyCreator,
          PropertyTag propertyTag,UInt32 propertySize,void *propertyBuffer);
OSStatus  GetWindowProperty(WindowRef window,PropertyCreator propertyCreator,
          PropertyTag propertyTag,UInt32 bufferSize,UInt32 *actualSize,void
*propertyBuffer);
OSStatus  GetWindowPropertySize(WindowRef window,PropertyCreator creator,PropertyTag tag,
          UInt32 *size);
OSStatus  RemoveWindowProperty(WindowRef window,PropertyCreator propertyCreator,
          PropertyTag propertyTag);
```

Adding to and Removing from the Update Region

```
OSStatus  InvalWindowRect(WindowRef window,const Rect *bounds);
OSStatus  InvalWindowRgn(WindowRef window,RgnHandle region);
OSStatus  ValidWindowRect(WindowRef window,const Rect *bounds);
OSStatus  ValidWindowRgn(WindowRef window,RgnHandle region);
```

Setting Content Region Color and Pattern

```
OSStatus  SetWindowContentColor(WindowRef window,RGBColor *color);
OSStatus  GetWindowContentColor(WindowRef window,RGBColor *color);
OSStatus  GetWindowContentPattern(WindowRef window,PixPatHandle outPixPat);
OSStatus  SetWindowContentPattern(WindowRef window,PixPatHandle pixPat);
```

Window Scrolling

```
OSStatus  ScrollWindowRect(WindowRef inWindow,const Rect *inScrollRect,SInt16 inHPixels,
          SInt16 inVPixels,ScrollWindowOptions inOptions,RgnHandle outExposedRgn);
OSStatus  ScrollWindowRegion(WindowRef inWindow,RgnHandle inScrollRgn,SInt16 inHPixels,
          SInt16 inVPixels,ScrollWindowOptions inOptions,RgnHandle outExposedRgn);
```

Creating a Window Menu

```
OSStatus  CreateStandardWindowMenu(OptionBits inOptions,MenuRef *outMenu);
OSStatus  SetWindowAlternateTitle(WindowRef inWindow,CFStringRef inTitle);
```

Demonstration Program Windows2 Listing

```
//
//****************************************************************************************
// Windows2.c                                                          CLASSIC EVENT
MODEL
//
//****************************************************************************************
//
// This program demonstrates the following Window Manager features and functions introduced
// with Mac OS 8.5:
//
// •  Creating floating windows and document windows using CreateNewWindow.
//
// •  Saving document windows and their associated data to a 'wind' resource.
//
// •  Creating document windows from 'wind' resources using CreateWindowFromResource.
//
// •  Managing windows in a floating windows environment.
//
// •  Setting and getting a window's property.
//
// •  Showing and hiding windows using TransitionWindow.
//
// •  Displaying window proxy icons.
//
// The program also demonstrates the creation of the system-managed Window menu introduced
// with Carbon and, on Mac OS X, a partial implementation of live window resizing.
//
// Those aspects of the newer Window Manager features not demonstrated in this program (full
// implementation of window proxy icons and window path pop-up menus) are demonstrated at
the
// demonstration program Files (Chapter 18).
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  An 'MBAR' resource, and 'MENU' resources for Apple, File, Edit, Document Windows and
//    Floating Windows menus (preload, non-purgeable).
//
// •  'TEXT' resources for the document windows (non-purgeable).
//
// •  'PICT' resources for the floating windows (non-purgeable).
//
// •  An 'ALRT' resource (purgeable), plus associated 'DITL', 'alrx', and 'dftb' resources
//    (all purgeable), for a movable modal alert invoked when the user chooses the About
//    Windows2... item from the  Apple/Application menu.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// In addition, the program itself creates a 'wind' resource, and saves it to the resource
// fork of the file titled "Document", when the user chooses CreateNewWindow from the
// Document Windows menu.
//
//
//****************************************************************************************

// ..........................................................................
includes
```

```
#include <Carbon.h>
// ..........................................................................
defines

#define rMenubar        128
#define mFile           129
#define  iQuit          12
#define rAboutAlert     128
#define rText           128
#define rColorsPicture 128
#define rToolsPicture   129
#define rWind           128
#define MIN(a,b)        ((a) < (b) ? (a) : (b))

// ..........................................................................
typedefs

typedef struct
{
  TEHandle editStrucHdl;
}  docStructure, **docStructureHandle;

// ..................................................................... global
variables

Boolean   gRunningOnX = false;
SInt16    gDocResFileRefNum;
WindowRef gColorsFloatingWindowRef;
WindowRef gToolsFloatingWindowRef;
Boolean   gDone;

// ..................................................................... function
prototypes

void    main                      (void);
void    doPreliminaries           (void);
OSErr   quitAppEventHandler       (AppleEvent *,AppleEvent *,SInt32);
void    doEvents                  (EventRecord *);
void    doMouseDown               (EventRecord *);
void    doUpdate                  (EventRecord *);
void    doUpdateDocumentWindow    (WindowRef);
void    doActivate                (EventRecord *);
void    doActivateDocumentWindow  (WindowRef,Boolean);
void    doOSEvent                 (EventRecord *);
void    doAdjustMenus             (void);
void    doMenuChoice              (SInt32);
OSErr   doCreateNewWindow         (void);
OSErr   doSaveWindow              (WindowRef);
OSErr   doCreateWindowFromResource (void);
OSErr   doCreateFloatingWindows   (void);
void    doCloseWindow             (WindowRef);
void    doErrorAlert              (SInt16);
void    doConcatPStrings          (Str255,Str255);

// ****************************************************************************
main

void  main(void)
{
  MenuBarHandle      menubarHdl;
  SInt32             response;
  MenuRef            menuRef;
  OSErr              osError;
  SInt16             numberOfItems, a;
  MenuCommand        menuCommandID;
  FSSpec             fileSpecTemp;
```

```
  EventRecord        eventStructure;
  SInt32             sleepTime;
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  UInt32             actualSize;

  // ..................................................................... do
preliminaries

  doPreliminaries();

  // ..................... set up menu bar and menus, customizes Window menu if running on
OS X

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    ExitToShell();
  SetMenuBar(menubarHdl);

  CreateStandardWindowMenu(0,&menuRef);
  InsertMenu(menuRef,0);

  DrawMenuBar();

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
      DisableMenuItem(menuRef,0);
    }
    gRunningOnX = true;
  }

  // ............................. open resource fork of file "Windows2 Document" and
store file reference number

  osError = FSMakeFSSpec(0,0,"\pWindows2 Document",&fileSpecTemp);
  if(osError == noErr)
    gDocResFileRefNum = FSpOpenResFile(&fileSpecTemp,fsWrPerm);
  else
    doErrorAlert(osError);

  // ..................................................................... create floating
windows

  if(osError = doCreateFloatingWindows())
    doErrorAlert(osError);

  // ..................................................................... enter
eventLoop

  gDone = false;
  sleepTime = GetCaretTime();

  while(!gDone)
  {
    if(WaitNextEvent(everyEvent,&eventStructure,sleepTime,NULL))
      doEvents(&eventStructure);
    else
    {
      if(eventStructure.what == nullEvent)
      {
```

```
        if(windowRef = FrontNonFloatingWindow())
        {
          if(!(GetWindowProperty(windowRef,0,'docs',sizeof(docStrucHdl),&actualSize,
                                 &docStrucHdl)))
            TEIdle((*docStrucHdl)->editStrucHdl);
        }
      }
    }
  }
}

// ****************************************************************************
doPreliminaries

void  doPreliminaries(void)
{
  OSErr osError;
  MoreMasterPointers(128);
  InitCursor();
  FlushEvents(everyEvent,0);
  osError = AEInstallEventHandler(kCoreEventClass,kAEQuitApplication,
                            NewAEEventHandlerUPP((AEEventHandlerProcPtr)
quitAppEventHandler),
                            0L,false);
  if(osError != noErr)
    ExitToShell();
}

// ****************************************************************************
doQuitAppEvent

OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  OSErr    osError;
  DescType returnedType;
  Size     actualSize;
  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,
NULL,0,
                              &actualSize);
  if(osError == errAEDescNotFound)
  {
    gDone = true;
    osError = noErr;
  }
  else if(osError == noErr)
    osError = errAEParamMissed;
  return osError;
}

// ****************************************************************************
doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
  switch(eventStrucPtr->what)
  {
    case mouseDown:
      doMouseDown(eventStrucPtr);
      break;

    case keyDown:
      if((eventStrucPtr->modifiers & cmdKey) != 0)
      {
        doAdjustMenus();
        doMenuChoice(MenuEvent(eventStrucPtr));
      }
```

```
      break;

    case updateEvt:
      doUpdate(eventStrucPtr);
      break;
    case activateEvt:
      doActivate(eventStrucPtr);
      break;

    case osEvt:
      doOSEvent(eventStrucPtr);
      break;
  }
}

// ********************************************************************************
doMouseDown

void  doMouseDown(EventRecord *eventStrucPtr)
{
  WindowRef      windowRef;
  WindowPartCode partCode, zoomPart;
  SInt32         menuChoice;
  WindowClass    windowClass;
  BitMap         screenBits;
  Rect           portRect, mainScreenRect;
  Point          standardStateHeightAndWidth;

  partCode = FindWindow(eventStrucPtr->where,&windowRef);

  switch(partCode)
  {
    case kHighLevelEvent:
      AEProcessAppleEvent(eventStrucPtr);
      break;

    case inMenuBar:
      doAdjustMenus();
      doMenuChoice(MenuSelect(eventStrucPtr->where));
      break;

    case inContent:
      GetWindowClass(windowRef,&windowClass);
      if(windowClass == kFloatingWindowClass)
      {
        if(windowRef != FrontWindow())
          SelectWindow(windowRef);
        else
        {
          if(windowRef == gColorsFloatingWindowRef)
            ; // Appropriate action for Colors floating window here.
          else if(windowRef == gToolsFloatingWindowRef)
            ; // Appropriate action for Tools floating window here.
        }
      }
      else
      {
        if(windowRef != FrontNonFloatingWindow())
          SelectWindow(windowRef);
        else
          ; // Appropriate action for active document window here.
      }
      break;
    case inDrag:
      DragWindow(windowRef,eventStrucPtr->where,NULL);
      break;
```

```
    case inGoAway:
      GetWindowClass(windowRef,&windowClass);
      if(windowClass == kFloatingWindowClass)
      {
        if(TrackGoAway(windowRef,eventStrucPtr->where) == true)
          TransitionWindow(windowRef,kWindowZoomTransitionEffect,
                           kWindowHideTransitionAction,NULL);
      }
      else
        if(TrackGoAway(windowRef,eventStrucPtr->where) == true)
          doCloseWindow(windowRef);
      break;

    case inGrow:
      ResizeWindow(windowRef,eventStrucPtr->where,NULL,NULL);
      GetWindowPortBounds(windowRef,&portRect);
      InvalWindowRect(windowRef,&portRect);
      break;

    case inZoomIn:
    case inZoomOut:
      mainScreenRect = GetQDGlobalsScreenBits(&screenBits)->bounds;
      standardStateHeightAndWidth.v = mainScreenRect.bottom - 100;
      standardStateHeightAndWidth.h = 600;

      if(IsWindowInStandardState(windowRef,&standardStateHeightAndWidth,NULL))
        zoomPart = inZoomIn;
      else
        zoomPart = inZoomOut;

      if(TrackBox(windowRef,eventStrucPtr->where,partCode))
        ZoomWindowIdeal(windowRef,zoomPart,&standardStateHeightAndWidth);
      break;
  }
}

// ********************************************************************************
doUpdate

void  doUpdate(EventRecord *eventStrucPtr)
{
  GrafPtr   oldPort;
  WindowRef windowRef;

  GetPort(&oldPort);
  windowRef = (WindowRef) eventStrucPtr->message;

  BeginUpdate(windowRef);

  SetPortWindowPort(windowRef);
  doUpdateDocumentWindow(windowRef);
  EndUpdate(windowRef);

  SetPort(oldPort);
}

// *****************************************************************
doUpdateDocumentWindow

void  doUpdateDocumentWindow(WindowRef windowRef)
{
  RgnHandle          visibleRegionHdl = NewRgn();
  Rect               contentRect;
  OSStatus           osError;
  UInt32             actualSize;
```

```
  docStructureHandle docStrucHdl;
  TEHandle           editStrucHdl;

   GetPortVisibleRegion(GetWindowPort(windowRef),visibleRegionHdl);
  EraseRgn(visibleRegionHdl);

  DrawGrowIcon(windowRef);

  if(!(osError = GetWindowProperty(windowRef,0,'docs',sizeof(docStrucHdl),&actualSize,
                                   &docStrucHdl)))
  {
    GetWindowPortBounds(windowRef,&contentRect);
    InsetRect(&contentRect,3,3);
    contentRect.right -= 15;
    contentRect.bottom -= 15;
    editStrucHdl = (*docStrucHdl)->editStrucHdl;
    (*editStrucHdl)->destRect = (*editStrucHdl)->viewRect = contentRect;
    TECalText(editStrucHdl);
    TEUpdate(&contentRect,(*docStrucHdl)->editStrucHdl);
  }
}

// *****************************************************************************
doActivate

void  doActivate(EventRecord *eventStrucPtr)
{
  WindowRef windowRef;
  Boolean   becomingActive;

  windowRef = (WindowRef) eventStrucPtr->message;
  becomingActive = ((eventStrucPtr->modifiers & activeFlag) == activeFlag);

  doActivateDocumentWindow(windowRef,becomingActive);
}

// *************************************************************
doActivateDocumentWindow

void  doActivateDocumentWindow(WindowRef windowRef,Boolean becomingActive)
{
  docStructureHandle docStrucHdl;
  UInt32             actualSize;
  OSStatus           osError;
  if(!(osError = GetWindowProperty(windowRef,0,'docs',sizeof(docStrucHdl),&actualSize,
                                   &docStrucHdl)))
  {
    if(becomingActive)
      TEActivate((*docStrucHdl)->editStrucHdl);
    else
      TEDeactivate((*docStrucHdl)->editStrucHdl);
  }
}

// *****************************************************************************
doOSEvent

void  doOSEvent(EventRecord *eventStrucPtr)
{
  switch((eventStrucPtr->message >> 24) & 0x000000FF)
  {
    case suspendResumeMessage:
      if((eventStrucPtr->message & resumeFlag) == 1)
        SetThemeCursor(kThemeArrowCursor);
      break;
  }
```

```
}

// ************************************************************************
doAdjustMenus

void  doAdjustMenus(void)
{
  MenuRef       floatMenuRef;
  Boolean       isVisible;
  MenuItemIndex menuItem;

  isVisible = IsWindowVisible(gColorsFloatingWindowRef);
  GetIndMenuItemWithCommandID(NULL,'fcol',1,&floatMenuRef,&menuItem);
  CheckMenuItem(floatMenuRef,menuItem,isVisible);

  isVisible = IsWindowVisible(gToolsFloatingWindowRef);
  GetIndMenuItemWithCommandID(NULL,'ftoo',1,&floatMenuRef,&menuItem);
  CheckMenuItem(floatMenuRef,menuItem,isVisible);

  DrawMenuBar();
}

// ************************************************************************
doMenuChoice

void  doMenuChoice(SInt32 menuChoice)
{
  MenuID        menuID;
  MenuItemIndex menuItem;
  OSErr         osError;
  MenuCommand   commandID;

  menuID = HiWord(menuChoice);
  menuItem = LoWord(menuChoice);
  if(menuID == 0)
    return;

  osError = GetMenuItemCommandID(GetMenuRef(menuID),menuItem,&commandID);
  if(osError == noErr && commandID != 0)
  {
    switch(commandID)
    {
      case 'abou':
        Alert(rAboutAlert,NULL);
        break;

      case 'quit':
        gDone = true;
        break;

      case 'cwin':
        if(osError = doCreateNewWindow())
          doErrorAlert(osError);
        break;

      case 'cwir':
        if(osError = doCreateWindowFromResource())
          doErrorAlert(osError);
        break;

      case 'fcol':
        if(IsWindowVisible(gColorsFloatingWindowRef))
          TransitionWindow(gColorsFloatingWindowRef,kWindowZoomTransitionEffect,
                           kWindowHideTransitionAction,NULL);
        else
          TransitionWindow(gColorsFloatingWindowRef,kWindowZoomTransitionEffect,
```

```
                          kWindowShowTransitionAction,NULL);
        break;

      case 'ftoo':
        if(IsWindowVisible(gToolsFloatingWindowRef))
          TransitionWindow(gToolsFloatingWindowRef,kWindowZoomTransitionEffect,
                          kWindowHideTransitionAction,NULL);
        else
          TransitionWindow(gToolsFloatingWindowRef,kWindowZoomTransitionEffect,
                          kWindowShowTransitionAction,NULL);
        break;
    }
  }

  HiliteMenu(0);
}

// **********************************************************************
doCreateFloatingWindows
OSErr  doCreateFloatingWindows(void)
{
  Rect      contentRect;
  OSStatus  osError;
  PicHandle pictureHdl;

  SetRect(&contentRect,102,59,391,132);

  if(!(osError = CreateNewWindow(kFloatingWindowClass,
                                 kWindowStandardFloatingAttributes |
                                 kWindowSideTitlebarAttribute,
                                 &contentRect,&gColorsFloatingWindowRef)))
  {
    if(pictureHdl = GetPicture(rColorsPicture))
      SetWindowPic(gColorsFloatingWindowRef,pictureHdl);
    osError = TransitionWindow(gColorsFloatingWindowRef,kWindowZoomTransitionEffect,
                               kWindowShowTransitionAction,NULL);
  }
  if(osError != noErr)
    return osError;

  SetRect(&contentRect,149,88,213,280);

  if(!(osError = CreateNewWindow(kFloatingWindowClass,
                                 kWindowStandardFloatingAttributes,
                                 &contentRect,&gToolsFloatingWindowRef)))
  {
    if(pictureHdl = GetPicture(rToolsPicture))
      SetWindowPic(gToolsFloatingWindowRef,pictureHdl);

    osError = TransitionWindow(gToolsFloatingWindowRef,kWindowZoomTransitionEffect,
                               kWindowShowTransitionAction,NULL);
  }

  return osError;
}

// **********************************************************************
doCreateNewWindow

OSErr  doCreateNewWindow(void)
{
  Rect               contentRect;
  OSStatus           osError;
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  Handle             textHdl;
```

```
  MenuRef            menuRef;

  SetRect(&contentRect,10,40,470,340);
  do
  {
    if(osError = CreateNewWindow(kDocumentWindowClass,kWindowStandardDocumentAttributes,
                                 &contentRect,&windowRef))
      break;

    if(gRunningOnX)
      ChangeWindowAttributes(windowRef,kWindowLiveResizeAttribute,0);

    if(!(docStrucHdl = (docStructureHandle) NewHandle(sizeof(docStructure))))
    {
      osError = MemError();
      break;
    }

    if(osError = SetWindowProperty(windowRef,0,'docs',sizeof(docStructure),
                                   &docStrucHdl))
      break;

    SetPortWindowPort(windowRef);
    UseThemeFont(kThemeSmallSystemFont,smSystemScript);

    textHdl = GetResource('TEXT',rText);
    osError = ResError();
    if(osError != noErr)
      break;

    OffsetRect(&contentRect,-contentRect.left,-contentRect.top);
    InsetRect(&contentRect,3,3);
    contentRect.right -= 15;
    contentRect.bottom -= 15;

    (*docStrucHdl)->editStrucHdl = TENew(&contentRect,&contentRect);
    TEInsert(*textHdl,GetHandleSize(textHdl),(*docStrucHdl)->editStrucHdl);

    SetWTitle(windowRef,"\pCreateNewWindow");

    if(osError = SetWindowProxyCreatorAndType(windowRef,0,'TEXT',kOnSystemDisk))
      break;
    if(osError = SetWindowModified(windowRef,false))
      break;
    if(osError = RepositionWindow(windowRef,NULL,kWindowCascadeOnMainScreen))
      break;
    if(osError = TransitionWindow(windowRef,kWindowZoomTransitionEffect,
                                  kWindowShowTransitionAction,NULL))
      break;

    if(osError = doSaveWindow(windowRef))
      break;

  }  while(false);
  if(osError)
  {
    if(windowRef)
      DisposeWindow(windowRef);
    if(docStrucHdl)
      DisposeHandle((Handle) docStrucHdl);
  }

  return osError;
}

// ************************************************************************
```

doSaveWindow

```
OSErr  doSaveWindow(WindowRef windowRef)
{
  SInt16             oldResFileRefNum;
  Collection         collection = NULL;
  OSStatus           osError;
  docStructureHandle docStrucHdl;
  UInt32             actualSize;
  Handle             flatCollectHdl, flatCollectResHdl, existingResHdl;

  oldResFileRefNum = CurResFile();
  UseResFile(gDocResFileRefNum);

  do
  {
    if(!(collection = NewCollection()))
    {
      osError = MemError();
      break;
    }

    if(osError = StoreWindowIntoCollection(windowRef,collection))
      break;

    if(osError = GetWindowProperty(windowRef,0,'docs',sizeof(docStrucHdl),&actualSize,
                                   &docStrucHdl))
      break;

    if(osError = AddCollectionItemHdl(collection,'TEXT',1,
                                      (*(*docStrucHdl)->editStrucHdl)->hText))
      break;

    if(!(flatCollectHdl = NewHandle(0)))
    {
      osError = MemError();
      break;
    }

    if(osError = FlattenCollectionToHdl(collection,flatCollectHdl))
      break;
    existingResHdl = Get1Resource('wind',rWind);
    osError = ResError();
    if(osError != noErr && osError != resNotFound)
      break;

    if(existingResHdl != NULL)
      RemoveResource(existingResHdl);
    osError = ResError();
    if(osError != noErr)
      break;

    AddResource(flatCollectHdl,'wind',rWind,"\p");
    osError = ResError();
    if(osError != noErr)
      break;

    flatCollectResHdl = flatCollectHdl;
    flatCollectHdl = NULL;

    WriteResource(flatCollectResHdl);
    osError = ResError();
    if(osError != noErr)
      break;

    UpdateResFile(gDocResFileRefNum);
```

```
    osError = ResError();
    if(osError != noErr)
      break;
  }  while(false);

  if(collection)
    DisposeCollection(collection);
  if(flatCollectHdl)
    DisposeHandle(flatCollectHdl);
  if(flatCollectResHdl)
    ReleaseResource(flatCollectResHdl);

  UseResFile(oldResFileRefNum);

  return osError;
}

// ************************************************************
doCreateWindowFromResource

OSErr  doCreateWindowFromResource(void)
{
  SInt16             oldResFileRefNum;
  OSStatus           osError;
  WindowRef          windowRef;
  Collection         unflattenedCollection = NULL;
  Handle             windResHdl;
  docStructureHandle docStrucHdl;
  SInt32             dataSize = 0;
  Handle             textHdl;
  Rect               contentRect;
  oldResFileRefNum = CurResFile();
  UseResFile(gDocResFileRefNum);

  do
  {
    if(osError = CreateWindowFromResource(rWind,&windowRef))
      break;

    if(gRunningOnX)
      ChangeWindowAttributes(windowRef,kWindowLiveResizeAttribute,0);

    if(!(unflattenedCollection = NewCollection()))
    {
      osError = MemError();
      break;
    }

    windResHdl = GetResource('wind',rWind);
    osError = ResError();
    if(osError != noErr)
      break;

    if(osError = UnflattenCollectionFromHdl(unflattenedCollection,windResHdl))
      break;

    if(!(docStrucHdl = (docStructureHandle) NewHandle(sizeof(docStructure))))
    {
      osError = MemError();
      break;
    }

    if(osError = GetCollectionItem(unflattenedCollection,'TEXT',1,&dataSize,
                                   kCollectionDontWantData))
      break;
```

```
    if(!(textHdl = NewHandle(dataSize)))
    {
      osError = MemError();
      break;
    }

    if(osError = GetCollectionItem(unflattenedCollection,'TEXT',1,kCollectionDontWantSize,
                                   *textHdl))
      break;

    GetWindowPortBounds(windowRef,&contentRect);
    contentRect.right -= 15;
    contentRect.bottom -= 15;
    SetPortWindowPort(windowRef);
    UseThemeFont(kThemeSmallSystemFont,smSystemScript);
    (*docStrucHdl)->editStrucHdl = TENew(&contentRect,&contentRect);
    TEInsert(*textHdl,dataSize,(*docStrucHdl)->editStrucHdl);

    if(osError = SetWindowProperty(windowRef,0,'docs',sizeof(docStrucHdl),&docStrucHdl))
      break;
    SetWTitle(windowRef,"\pCreateWindowFromResource");

    if(osError = SetWindowProxyCreatorAndType(windowRef,0,'TEXT',kOnSystemDisk))
      break;
    if(osError = SetWindowModified(windowRef,false))
      break;
    if(osError = RepositionWindow(windowRef,NULL,kWindowCascadeOnMainScreen))
      break;
    if(osError = TransitionWindow(windowRef,kWindowZoomTransitionEffect,
                                  kWindowShowTransitionAction,NULL))
      break;
  }  while(false);

  if(unflattenedCollection)
    DisposeCollection(unflattenedCollection);
  if(windResHdl)
    ReleaseResource(windResHdl);

  UseResFile(oldResFileRefNum);

  return osError;
}

// ****************************************************************************
doCloseWindow

void  doCloseWindow(WindowRef windowRef)
{
  OSStatus           osError;
  docStructureHandle docStrucHdl;
  UInt32             actualSize;

  do
  {
    if(osError = TransitionWindow(windowRef,kWindowZoomTransitionEffect,
                                  kWindowHideTransitionAction,NULL))
    break;

    if(osError = GetWindowProperty(windowRef,0,'docs',sizeof(docStrucHdl),&actualSize,
                                   &docStrucHdl))
    break;
  }  while(false);

  if(osError)
    doErrorAlert(osError);
```

```
  if((*docStrucHdl)->editStrucHdl)
    TEDispose((*docStrucHdl)->editStrucHdl);

  if(docStrucHdl)
    DisposeHandle((Handle) docStrucHdl);

  DisposeWindow(windowRef);
}
// ************************************************************************
doErrorAlert

void  doErrorAlert(SInt16 errorCode)
{
  Str255 errorCodeString;
  Str255 theString = "\pAn error occurred. The error code is ";
  SInt16 itemHit;

  NumToString((SInt32) errorCode,errorCodeString);
  doConcatPStrings(theString,errorCodeString);

  StandardAlert(kAlertStopAlert,theString,NULL,NULL,&itemHit);
  ExitToShell();
}

// ************************************************************************
doConcatPStrings

void  doConcatPStrings(Str255 targetString,Str255 appendString)
{
  SInt16  appendLength;

  appendLength = MIN(appendString[0],255 - targetString[0]);

  if(appendLength > 0)
  {
    BlockMoveData(appendString+1,targetString+targetString[0]+1,(SInt32) appendLength);
    targetString[0] += appendLength;
  }
}

//
********************************************************************************
```

Demonstration Program Windows2 Comments

Two Window Manager features introduced with Mac OS 8.5 (full window proxy icon implementation and window path pop-up menus) are not demonstrated in this program. However, they are demonstrated at the demonstration program associated with Chapter 18.

When the program is run, the user should

- Choose CreateNewWindow from the Document Windows menu, noting that, when the new window is displayed, the floating windows and the new (document) window are all active.

  (Note: As well as creating the window, the program loads and displays a 'TEXT' resource (simulating a document associated with the window) and then saves the window and the text to a 'wind' resource.)

- Choose CreateWindowFromResource from the Document Windows menu, noting that the window is created from the 'wind' resource saved when CreateNewWindow was chosen.

- Choose About Windows2… from the Apple menu, noting that the floating windows appear in the deactivated state when the alert box opens.

- Hide the floating windows by clicking their close boxes, and toggle the floating windows between hidden and showing by choosing their items in the Floating Windows menu, noting the transitional animations.

- Click in the Finder to send the application to the background, noting that the floating windows are hidden by this action. Then click in one of the application's windows, noting that the floating windows re-appear.

- Note the transitional animations when the document windows are opened and closed.

- Exercise the system-managed Window menu, noting the customization of this menu when the program is run on Mac OS X.

defines

`rWind` represents the ID of the `'wind'` resource created by the program.

typedefs

A document structure of type `docStructure` will be associated with each document window. The single field in the document structure (`editStrucHdl`) will be assigned a handle to a TextEdit edit structure, which will contain the text displayed in the window.

Global Variables

`gDocResFileRefNum` will be assigned the file reference number for the resource fork of the document file "Windows2 Document" included in the demo program's folder. `gColorsFloatingWindowRef` and `gToolsFloatingWindowRef` will be assigned references to the window objects for the floating windows.

main

The call to `CreateStandardWindowMenu` creates the system-managed Window menu, which is added to the menu list by the call to `InsertMenu`. If the program is running on Mac OS X, the next block customizess the Window menu by searching for the item with the command ID 'wldv' (that is, the divider between the commands and the individual window items), inserting a divider and two custom items before that item, and assigning command IDs to those items. (At the time of writing, the divider did not have the 'wldv' command ID in CarbonLib.)

The resource fork of the file titled "Windows2 Document" is opened and the file reference number is saved to a global variable. The program will be saving a 'wind' resource to this file's resource fork.

`CurResFile` is called to set the application's resource fork as the current resource file.

The function `doCreateFloatingWindows` is called to create and show the floating windows.

In the next block (the main event loop), `WaitNextEvent`'s sleep parameter is assigned the value returned by `GetCaretTime`. (`GetCaretTime` returns the value stored in the low memory global `CaretTime`, which determines the blinking rate for the insertion point caret as set by the user. This ensures that `TEIdle`, which causes the caret to blink, will be called at the correct interval.

When `WaitNextEvent` returns 0 with a null event, `FrontNonFloatingWindow` is called to obtain a reference to the front document window. If such a window exists, `GetWindowProperty` is called to retrieve a handle to the window's document structure. The handle to the TextEdit edit structure, which is stored in the window's document structure, is then passed in the call to `TEIdle`, which causes the insertion point caret to

blink.

doMouseDown

`doMouseDown` continues the processing of mouse-down events, switching according to the part code.

The `inContent` case is handled differently depending on whether the event is in a floating window or a document window. `GetWindowClass` returns the window's class. If the window is a floating window, and if that window is not the front floating window, `SelectWindow` is called to bring that floating window to the front. If the window is the front floating window, the identity of the window is determined and the appropriate further action is taken. (In this demonstration, no further action is taken.)

If the window is not a floating window, and if the window is not the front non-floating window, `SelectWindow` is called to

- Unhighlight the currently active non-floating window, bring the specified window to the front of the non-floating windows, and highlight it.
- Generate activate events for the two windows.
- Move the previously active non-floating window to a position immediately behind the specified window.

If the window is the front non-floating window, the appropriate further action is taken. (In this demonstration, no further action is taken.)

The `inGoAway` case is also handled differently depending on whether the event is in a floating window or a document window. `TrackGoAway` is called in both cases to track user action while the mouse-button remains down. If the pointer is still within the go away box when the mouse-button is released, and if the window is a floating window, `TransitionWindow` is called to hide the window. If the window is a non-floating window, the function `doCloseWindow` is called to close the window.

doUpdate

`doUpdate` further processes update events. When an update event is received, doUpdate calls `doUpdateDocumentWindow`. (As will be seen, in this particular demonstration, the Window Manager will not generate updates for the floating windows.)

doUpdateDocumentWindow

`doUpdateDocumentWindow` is concerned with the drawing of the content region of the non-floating windows.

`GetWindowProperty` is then called to retrieve the handle to the window's document structure, which, as previously stated, contains a handle to a TextEdit structure containing the text displayed in the window. If the call is successful, measures are taken to redraw the text in the window, taking account of the current height and width of the content region less the area that would ordinarily be occupied by scroll bars. (The TextEdit calls in this section are incidental to the demonstration. TextEdit is addressed in Chapter 21.)

doActivateDocumentWindow

`doActivateDocumentWindow` performs, for the non-floating windows, those window activation actions for which the application is responsible. In this demonstration, that action is limited to calling `TEActivate` or `TEDeactivate` to show or remove the insertion point caret.

`GetWindowProperty` is called to retrieve the handle to the window's document structure, which contains a handle to the TextEdit structure containing the text displayed in the window. If this call is successful, and if the window is being activated, `TEActivate` is called to display the insertion point caret. If the window is being deactivated, `TEDeactivate` is called to remove the insertion point caret.

doAdjustMenus

`doAdjustMenus` is called in the event of a mouse-down event in the menu bar when a key is pressed together with the Command key. The function checks or unchecks the items in the Floating Windows menu depending on whether the associated floating window is currently showing or hidden.

doMenuChoice

`doMenuChoice` switches according to the menu choices of the user.

If the user chooses the About Windows2… item from the Apple menu, Alert is called to display the About Windows2… alert box.

If the user chose the first item in the Document Windows menu, the function `doCreateNewWindow` is called. If the user chose the second item, the function `doCreateWindowFromResource` is called. If either of these functions return an error, an error-handling function is called.

When an item in the Floating Windows menu is chosen, `IsWindowVisible` is called to determine the visibility state of the relevant floating window. `TransitionWindow` is

then called, with the appropriate constant passed in the action parameter, to hide or show the window depending on the previously determined current visibility state.

doCreateFloatingWindows

`doCreateFloatingWindows` is called from main to create the floating windows.

The Colors floating window is created first. `SetRect` is called to define a rectangle, which will be used to establish the size of the window and its opening location in global coordinates. `CreateNewWindow` is then called to create a floating window (first parameter) with a close box, a collapse box, and a side title bar (second parameter), and with the previously defined content region size and location (third parameter).

If this call is successful, `GetPicture` is called to load the specified `'PICT'` resource. If the resource is loaded successfully, `SetWindowPic` is called to store the handle to the picture structure in the `windowPic` field of the window's color window structure. This latter means that the Window Manager will draw the picture in the window instead of generating update events for it. Finally, `TransitionWindow` is called to make the window visible (with animation and sound).

The same general procedure is then followed to create the Tools floating window.

doCreateNewWindow

`doCreateNewWindow` is called when the user chooses Create New Window from the Document Windows menu. In addition to creating a window, and for the purposes of this demonstration, `doCreateNewWindow` also saves the window and its associated data (text) in a 'wind' resource.

`SetRect` is called to define a rectangle that will be used to establish the size of the window and its opening location in global coordinates. The call to `CreateNewWindow` creates a document window (first parameter) with a close box, a full zoom box, a collapse box, and a size box (second parameter), and with the previously defined content region size and location (third parameter).

If the program is running on Mac OS X, `ChangeWindowAttributes` is called to set the `kWindowLiveResizeAttribute`. This results in a partial implementation of live resizing.

`NewHandle` is then called to create a relocatable block for the document structure to be associated with the window. `SetWindowProperty` associates the document structure with the window. 0 is passed in the `propertyCreator` parameter because this demonstration has no application signature. The value passed in the `propertyTag` parameter ('docs') is just a convenient value with which to identify the data.

The call to `SetPortWindowPort` sets the window's graphics port as the current port and the call to `UseThemeFont` sets the window's font to the small system font.

The next three blocks load a `'TEXT'` resource, insert the text into a TextEdit structure, and assign a handle to that structure to the editStrucHdl field of the window's document structure. This is all for the purpose of simulating some text that the user has typed into the window.

`SetWTitle` sets the window's title.

The window lacks an associated file, so `SetWindowProxyCreatorAndType` is called to cause a proxy icon to be displayed in the window's drag bar. 0 passed in the `fileCreator` parameter and `'TEXT'` passed in the `fileType` parameter cause the system's default icon for a document file to be displayed. `SetWindowModified` is then called with false passed in the modified parameter to cause the proxy icon to appear in the enabled state (indicating no unsaved changes).

The call to `RepositionWindow` positions the window relative to other windows according to the constant passed in the method parameter.

As the final step in creating the window, `TransitionWindow` is called to make the window visible (with animation).

To facilitate the demonstration of creating a window from a `'wind'` resource (see the function `doCreateWindowFromResource`), a function is called to save the window and its data (the text) to a 'wind' resource in the application's resource fork.

If an error occurred within the do/while loop, if a window was created, it is disposed of. Also, if a nonrelocatable block for the document structure was created, it is disposed of.

doSaveWindow

`doSaveWindow` is called by `doCreateNewWindow` to save the window and its data (the text) to a `'wind'` resource.

The current resource file's file reference number is saved and the resource fork of the document titled "Windows2 Document" is made the current resource file.

The call to the Collection Manager NewCollection function allocates memory for a new collection object and initializes it. The call to `StoreWindowIntoCollection` stores data describing the window into the collection.

`GetWindowProperty` retrieves the handle to the window's document structure.

The handle to the window's text is stored in the `hText` field of the TextEdit structure. The handle to the TextEdit structure is, in turn, stored in the window's document structure. The Collection Manager function `AddCollectionItemHdl` adds a new item to the collection, specifically, a copy of the text.

The call to `NewHandle` allocates a zero-length handle, which will be used to hold a flattened collection. The Collection Manager function `FlattenCollectionToHdl` flattens the collection into a Memory Manager handle.

The next six blocks use Resource Manager functions to save the flattened collection as a `'wind'` resource in the resource fork of the application file.

`Get1Resource` attempts to load a `'wind'` resource with ID 128. If `ResError` reports an error, and if the error is not the "resource not found" error, the whole save process is aborted. (Accepting the "resource not found" error as an acceptable error caters for the possibility that this may be the first time the window and its data have been saved.)

If `Get1Resource` successfully loaded a `'wind'` resource with ID 128, `RemoveResource` is called to remove that resource from the resource map, `AddResource` is called to make the flattened collection in memory into a `'wind'` resource, assigning a resource type, ID and name to that resource, and inserting an entry in the resource map for the current resource file. `WriteResource` is called to write the resource to the document file's resource fork. Since the resource map has been changed, `UpdateResFile` is called to update the resource map on disk.

Below the do/while loop, the collection and the flattened collection block are disposed of and the resource in memory is released.

Finally, the saved resource file is made the current resource file.

doCreateWindowFromResource

`doCreateWindowFromResource` creates a window from the `'wind'` resource created by `doSaveWindow`.

The current resource file's file reference number is saved and the resource fork of the document titled "Windows2 Document" is made the current resource file.

`CreateWindowFromResource` creates a window, invisibly, from the `'wind'` resource with ID 128.

The call to the Collection Manager `NewCollection` function creates a new collection.

`GetResource` loads the `'wind'` resource with ID 128. The Collection Manager function `UnflattenCollectionFromHdl` unflattens the 'wind' resource and stores the unflattened collection in the collection object `unflattenedCollection`.

`NewHandle` allocates a relocatable block the size of a window document structure.

The Collection Manager function `GetCollectionItem` is called twice, the first time to get the size of the text data, not the data itself. (The item in the collection is specified by the second and third parameters (tag and ID)). This allows the call to `NewHandle` to create a relocatable block of the same size. `GetCollection` is then called again, this time to obtain a copy of the text itself.

The next block creates a new TextEdit structure (`TENew`), assigning its handle to the `editStrucHdl` field of the document structure, which will be associated with the window. `TEInsert` inserts the copy of the text obtained by the second call to `GetCollectionItem` into the `TextEdit` structure.

The call to `SetWindowProperty` associates the document structure with the window, thus associating the TextEdit structure and its text with the window.

`SetWTitle` sets the window's title.

The window lacks an associated file, so the Mac OS 8.5 function `SetWindowProxyCreatorAndType` is called to cause a proxy icon to be displayed in the window's drag bar. 0 passed in the fileCreator parameter and `'TEXT'` passed in the `fileType` parameter cause the system's default icon for a document file to be displayed. `SetWindowModified` is then called with false passed in the modified parameter to cause the proxy icon to appear in the enabled state (indicating no unsaved changes).

The call to `RepositionWindow` positions the window relative to other windows according to the constant passed in the method parameter.

As the final step in creating the window, `TransitionWindow` is called to make the window visible (with animation).

Below the do/while loop, the unflattened collection is disposed of and the `'wind'` resource is released.

Finally, the saved resource file is made the current resource file.

doCloseWindow

`doCloseWindow` is called when the user clicks the close box of a document window.

`TransitionWindow` is called to hide the window (with animation). `GetWindowProperty` is then called to retrieve a handle to the window's document structure, allowing the memory occupied by the TextEdit structure and document structure associated with the window to be disposed of. `DisposeWindow` is then called to remove the window from the window list and discard all its data storage.

doErrorAlert

`doErrorAlert` is called when errors are detected. In this demonstration, the action taken is somewhat rudimentary. A stop alert box displaying the error number is invoked. When the user dismisses the alert box, the program terminates. `eventFilter` supports `doErrorAlert`.

Chapter 17. The Carbon Event Manager

IN THIS CHAPTER

- Overview
- Event Reference, Class, and Type
- Standard Event Handlers
- The Application's Event Handlers
- Sending and Explicitly Propagating Events
- Event Timers
- Getting Event Time
- Other Aspects of the Carbon Event Model
- Main Constants, Data Types, and Functions
- Relevant Window Manager Constants and Functions
- Demonstration Program CarbonEvents1 Listing
- Demonstration Program CarbonEvents1 Comments
- Demonstration Program CarbonEvents2 Listing
- Demonstration Program CarbonEvents2 Comments

Overview

The Carbon Event Model

The Carbon event model, which was introduced with Carbon as an alternative to what is now termed the Classic event model, reduces the amount of events-related code required by an application and, in addition, facilitates a more efficient allocation of processing time on the preemptive multitasking Mac OS X. Indeed, the Carbon event model is the underlying event model on Mac OS X, the Classic event model being constructed on top of this model and emulated by the Carbon Event Manager.

Event Handling Basics

As will by now be apparent, applications using the Classic event model spend a large amount of time in the `WaitNextEvent` loop handling such user-interface events as mouse-downs and key-downs. In the Carbon event model, this continual and inefficient "polling" for events is avoided, events being dispatched directly to Toolbox objects.

Standard Event Handlers

These dispatched events may be handled automatically by **standard (default) event handlers** provided by the Carbon Event Manager if you so specify. The provision of standard event handlers greatly simplifies the programming task. As an example, and as will be seen in the demonstration program CarbonEvents1, your application requires no code at all to handle basic window dragging, resizing, zooming, activation, and deactivation operations.

Standard event handlers are provided for each type of **event target** (windows, menus, controls, and the application itself).

Overriding and Complementing the Standard Handlers

At the same time, you can override or complement the standard behavior provided by the standard event handlers by writing your own handlers and installing them on the relevant objects. Your application's event handler will override the standard event handler if it returns `noErr`, which defeats the event being passed to the standard handler. Your application's event handler will complement the standard event handler if it returns `eventNotHandledErr`, which causes the event to be further propagated to the standard event handler following handling by your application's event handler. (Event handlers are installed on a stack, the most recently installed on top. The most recently installed handler is called first.)

The Basic Approach

The basic approach to using the Carbon event model API is thus to install the relevant standard event handlers first and then register the types of events your application wishes to receive in order to override or complement the actions of the standard handlers.

Event Propagation Order

Events are propagated in a particular order, that order depending on the type of event. For example, control-related events are sent first to the control, then to the owning window, and then to the application. This means that you can install a handler for the control on either the control, the owning window (as in the demonstration program CarbonEvents1), or the application.

As another example, menu-related events are sent first to the menu, then to the user focus target (that is, the object with current keyboard focus, which can be either a window or a control), then to the application.

RunApplicationEventLoop

At the point where a Classic event model version of your application would call `WaitNextEvent` to enter the main event loop, your Carbon event model application calls the Carbon Event Manager function `RunApplicationEventLoop`. `RunApplicationEventLoop`

- Moves events from lower-level OS queues into the Carbon queue.
- Dispatches those events from the Carbon queue to the standard event handlers and, for events types that your application has registered, to your application's event handlers.

When an event occurs that requires your program's attention, the Carbon Event Manager calls the handler for that event type. On return from the handler, your program is suspended until the next event requiring its attention is received. Thus your program only uses processor time when processing an event, other programs and processes running in the meantime.

Event Timers

The Carbon Event Manager supports the installation of **timers,** which can be set to fire either once or repeatedly, and which may be used to trigger calls to a specified function at a specified elapsed time or at specified intervals.

Event Reference, Class, and Type

Event Reference

The **event reference** is the core Carbon Event Manager data structure:

```
typedef struct OpaqueEventRef *EventRef;
```

Event Class and Type

As was stated in Chapter 2, the Classic event model is limited to a maximum of 16 event types. By contrast, the Carbon event model can accommodate an unlimited number of **event types.** Event types are grouped by **event class.**

Typical event classes, as represented by constants in CarbonEvents.h, are as follows:

```
kEventClassApplication
kEventClassWindow
kEventClassControl
kEventClassMenu
```

Each event class comprises a number of event types. For example, some of the many event types pertaining to the `kEventClassWindow` event class, as represented by constants in CarbonEvents.h, are as follows:

```
kEventWindowDrawContent
kEventWindowActivated
kEventWindowClickDragRgn
kEventWindowGetIdealSize
```

Given an event reference, your application can ascertain the class and type of a received event by calling `GetEventClass` and `GetEventKind`.

Standard Event Handlers

Standard Application Event Handler

The standard application event handler is installed automatically when your application calls `RunApplicationEventLoop`. Amongst other things, the standard application event handler handles application-activated and application-deactivated events (in Classic event model parlance, resume and suspend events).

Standard Window Event Handler

The standard window event handler handles all of the possible user interactions with a window (dragging, resizing, zooming, activation, deactivation, and so on.). It must be explicitly installed on the target window by your application. You can cause the standard window event handler to be installed on a window as follows:

- For a window created from a `'WIND'` resource using `GetNewCWindow`, either set the standard handler attribute in a call to the function `ChangeWindowAttributes`, for example:

  ```
  ChangeWindowAttributes(windowRef,kWindowStandardHandlerAttribute,0);
  ```

  or call `InstallStandardEventHandler`, passing in an event target reference (type `EventTargetRef`) obtained using `GetWindowEventTarget`, for example:

  ```
  InstallStandardEventHandler(GetWindowEventTarget(windowRef));
  ```

- For a window created using CreateNewWindow, pass kWindowStandardHandlerAttribute in the attributes parameter.

The Application's Event Handlers

Handlers are Callback Functions

The handlers provided by your application are callback functions. When called, they are passed

- A reference to the event handler call (type `EventHandlerCallRef`).
- The event reference, from which you can extract the event class and type.
- A pointer to user data (assuming that you passed a pointer to that data when you installed the handler).

Installing the Application's Event Handlers

You can install handlers provided by your application using `InstallEventHandler`:

```
OSStatus  InstallEventHandler(EventTargetRef inTarget,EventHandlerUPP inHandler,
                              UInt32 inNumTypes,const EventTypeSpec *inList,
                              void *inUserData,EventHandlerRef *outRef);
```

`inTarget` An event target reference to the event target the handler is to be registered with. Use one of the following functions to obtain this reference:

```
GetApplicationEventTarget
GetWindowEventTarget
GetControlEventTarget
GetMenuEventTarget
GetUserFocusEventTarget
```

`inHandler` A universal procedure pointer to the handler function provided by your application.

`inNumTypes` The number of event types to be registered by this call to `InstallEventHandler`.

`inList` A pointer to an array of type `EventTypeSpec` structures specifying the event types being registered. The `EventTypeSpec` structure is as follows:

```
struct EventTypeSpec
{
  UInt32 eventClass;  // Event class
  UInt32 eventKind;   // Event type
} ;
typedef struct EventTypeSpec EventTypeSpec;
```

For example, if you wished to register the `kEventWindowDrawContent` and `kEventWindowActivated` event types, you would define the array as follows:

```
EventTypeSpec myTypes[] = {  {  kEventClassWindow, kEventWindowDrawContent } ,
                             {  kEventClassWindow, kEventWindowActivated   }  } ;
```

`inUserData` Optionally, a pointer to data to be passed to your event handler when it is called.

`outRef` If you will later need to remove the handler, pass a pointer to a variable of type `EventHandlerRef` in this parameter. On return, this variable will receive the event handler reference that will be required by your call to `RemoveEventHandler`.

You can also use the following macros, which are derived from the function `InstallEventHandler`, to install your application's handlers:

```
InstallApplicationEventHandler
InstallWindowEventHandler
InstallControlEventHandler
InstallMenuEventHandler
```

Different objects of the same type do not have to have the same handler. For example, you can install separate handlers on each of two windows.

Inside The Application's Event Handlers

Getting Event Parameters

In some circumstances, in order to correctly handle a particular event type, you may need to extract specific data from the event using the function `GetEventParameter`. For example, on receipt of an event in the `kEventClassWindow` class, you will almost invariably need to call `GetEventParameter` to get the window reference required to facilitate the handling of certain event types in that class. Similarly, on receipt of an event of type `kEventMouseDown`, you might need to call `GetEventParameter` to obtain the mouse location.

The `GetEventParameter` prototype is as follows:

```
OSStatus  GetEventParameter(EventRef inEvent,EventParamName inName,
                            EventParamType inDesiredType,EventParamType *outActualType,
                            UInt32 inBufferSize,UInt32 *outActualSize,void *outData);
```

`inEvent` A reference to the event.

`inName` The parameter's symbolic name. Symbolic names pertaining to the various event types are listed in CarbonEvents.h immediately after the enumerations for those types. For example, the symbolic name for the mouse location is `kEventParamMouseLocation`.

`inDesiredType` The type of data. This is listed against the parameter's symbolic name in CarbonEvents.h. For example, the type of data pertaining to the symbolic name `kEventParamMouseLocation` is `typeQDPoint`.

`outActualType` Actual type of value returned. Specify `NULL` if this information is not needed.

`inBufferSize` The size of the output buffer.

`outActualSize` Actual size of value returned. Specify `NULL` if this information is not needed.

`outData` A pointer to the buffer that will receive the parameter data.

The types of data that can be extracted from an event depends on the type of event. The parameter symbolic names and data types listed in CarbonEvents.h together indicate the type, or types, of data obtainable from an event of a particular type.

Event Parameters and Command Events

The Carbon Event Manager can associate special **command events** with menu items with command IDs. You can, of course, assign your own command IDs to menu items using `SetMenuItemCommandID`; however, note that CarbonEvents.h defines command IDs for many common menu items, for example:

```
kHICommandQuit  = FOUR_CHAR_CODE('quit')
kHICommandUndo  = FOUR_CHAR_CODE('undo')
kHICommandCut   = FOUR_CHAR_CODE('cut ')
kHICommandPaste = FOUR_CHAR_CODE('past')
```

When a menu item with a command ID is chosen by the user, either with the mouse or using a Command-key equivalent, the Carbon Event Manager dispatches the relevant command event (class `kEventClassCommand`, type

kEventProcessCommand).

When your application's handler receives a kEventProcessCommand event type, you pass kEventParamDirectObject in the inName parameter of your GetEventParameter call, typeHICommand in the inDesiredType parameter, and the address of a structure of type HICommand in the outData parameter. The HICommand structure is as follows:

```
struct HICommand
{
  UInt32 attributes;
  UInt32 commandID;
  struct
  {
    MenuRef menuRef;
    UInt16  menuItemIndex;
  }  menu;
} ;
typedef struct HICommand HICommand;
```

Thus you will be able to extract the menu reference and menu item number, as well as the command ID of the chosen menu item (if any), from the data returned by the call to GetEventParameter.

Quit Command Handling

The Quit command event is a special case. When the Quit item is chosen, the standard application event handler calls either the default Quit Application Apple event handler or your application's Quit Application Apple event handler if it has installed its own. (When your application calls RunApplicationEventLoop, the default Quit Application Apple event handler is automatically installed if the application has not installed its own.)

Thus the only action required by your application's handler is to ensure that it returns eventNotHandledErr when it determines that the commandID field of the HICommand structure contains kHICommandQuit, thereby causing the event to be propagated to the standard application event handler and thence to the relevant Quit Application Apple event handler.

For this to work on Mac OS 8/9, your application must assign the command ID kHICommandQuit to the Quit item at program start when the application determines that it is running on Mac OS 8/9.

Setting Event Parameters

In certain circumstances, your handler will need to call SetEventParameter to set a piece of data for a given event. For example, suppose you wish to constrain window resizing to a specified minimum size and, accordingly, register for the kEventWindowGetMinimumSize event type. When this event type is received by your handler (it will be dispatched when a mouse-down occurs in the size box/resize control of a window on which your handler is installed), your handler should call SetEventParameter with kEventParamDimensions passed in the inName parameter and a pointer to a variable of type Point passed in the inDataPtr parameter. (The Point variable should contain the desired minimum window height and width.)

Converting an Event Reference to an Event Record

In certain circumstances, your handler may need to convert the event reference to a Classic event model event structure (type EventRecord) in order to be able to handle the event. You can use the function ConvertEventRefToEventRecord for that purpose.

Menu Adjustment

You can ensure that your application's menu adjustment function is called when appropriate by registering the kEventMenuEnableItems event type (kEventClassMenu event class) and calling your menu adjustment function when that event type is received. The kEventMenuEnableItems event type will be dispatched when a mouse-down occurs in the menu bar and when a menu-related Command-key equivalent is pressed.

Cursor Shape Changing

In Classic event model applications, the application's cursor shape-changing function is typically called when mouse-moved Operating System events are received. An alternative "trigger" is required when using the Carbon event model.

One approach is to install a Carbon events timer set to fire at an appropriate interval and call the cursor shape-changing function when the timer fires. However, this method is not recommended for Mac OS X because it is somewhat like polling for an event, which is more processor-intensive.

The recommended approach is to register for the `kEventMouseMoved` event type (`kEventClassMouse` event class) and call the cursor shape-changing function on receipt of that event type.

Window Updating

To accommodate window content region updating (re-drawing) requirements, your application should register for the `kEventWindowDrawContent` event type (`kEventClassWindow` event class) and call its update function when that event type is received.

Note that the Window Manager sends an event of type `kEventWindowUpdate` to all windows that need updating, regardless of whether those windows have the standard window event handler installed. If the standard window event handler is installed, then when the standard handler receives the `kEventWindowUpdate` event, it calls `BeginUpdate`, sends a `kEventWindowDrawContent` event, and calls `EndUpdate`. There is thus no need for your update function to call `BeginUpdate` and `EndUpdate` when responding to `kEventWindowDrawContent` events.

Handler Disposal

All handlers on a target are automatically disposed of when the target is disposed of.

Sending and Explicitly Propagating Events

Sending Events

You can send an event to a specified event target using either the function `SendEventToEventTarget` or the following macros derived from that function:

`SendEventToApplication`
`SendEventToWindow`
`SendEventToControl`
`SendEventToMenu`
`SendEventToUserFocus`

Explicitly Propagating Events

You can explicitly propagate an event up the propagation chain by calling `CallNextEventHandler` within your event handler. This is useful in circumstances where, for example, you wish to incorporate the standard handler's response into your own pre- or post-processing.

Event Timers

Event timers may be used for many purposes. The most common one is to trigger a call to your application's idle function, perhaps for the purpose of blinking the insertion point caret. You can use `InstallEventLoopTimer` to install an event timer:

```
OSStatus  InstallEventLoopTimer(EventLoopRef inEventLoop,EventTimerInterval inFireDelay,
                                EventTimerInterval inInterval,
                                EventLoopTimerUPP inTimerProc, void *inTimerData,
                                EventLoopTimerRef *outTimer);
```

`inEventLoop` The event loop on which the timer is to be installed. Usually, this will be the event loop reference returned by a call to `GetCurrentEventLoop`.

`inFireDelay` The required delay before the timer first fires. This can be `0`.

`inInterval` A value of type `double` specifying the interval at which the timer is required to fire. For one-shot timers, `0` should be passed in this parameter. For a timer whose purpose is to trigger calls to an idle function which blinks the insertion point caret, pass the value returned by a call to `GetCaretTime` converted to **event time** by the macro `TicksToEventTime`.

Note that event time is in seconds since system startup. You can use the macros `TicksToEventTime` and `EventTimeToTicks` to convert between ticks and event time.

`inTimerProc` A universal procedure pointer to the function to be called when the timer fires.

`inTimerData` Optionally, a pointer to data to be passed to the function called when the timer fires.

`OutTimer` A reference to the newly installed timer. Usually, this will be required only if you intend to remove the timer at some point.

Note that, depending on the parameters passed to this function, the timer can be set to fire either once or repeatedly at a specified interval.

You can remove an installed timer by calling `RemoveEventLoopTimer`.

Getting Event Time

Your application can determine the time an event occurred using the function `GetEventTime`. It can also determine the time from system startup using the function `GetCurrentEventTime`.

Other Aspects of the Carbon Event Model

The Carbon Event Model and Apple Events

Your application requires no code at all to ensure that, when Apple events are dispatched to it, its Apple event handlers are called.

Carbon Event Model and Control Hierarchies

When you establish an embedding hierarchy for controls, you are also establishing an event handling chain. When you click in a given control, the event is sent first to that control. If that control does not handle the event (that is, its handler returns `eventNotHandledErr`), the event is passed up the chain to the control that contains the first control, and so on up the chain.

Carbon Event Model and Event Filter Functions

In Classic event model applications, you must pass a universal procedure pointer to an application-defined event filter (callback) function in the `modalFilter` parameter of `ModalDialog`, and call your application's window updating function within the filter function.

Calling your window updating function from within your event filter function is not necessary in Carbon event model applications provided the application installs the standard window event handler on the relevant windows, registers for the `kEventWindowDrawContent` event type, and calls its window updating function when that event type is received.

Mouse Tracking

The demonstration program QuickDraw (Chapter 12) uses the Event Manager function `StillDown` in the `doDrawWithMouse` function to determine whether the mouse button has been continuously pressed since the most recent `mouseDown` event. The Event Manager function `WaitMouseUp` is often used for similar purposes.

For reasons of efficient use of processor cycles, `TrackMouseLocation` should be used in lieu of `StillDown` and `WaitMouseUp` in applications intended to run on Mac OS X. (`TrackMouseLocation` does not return control to your application until the mouse is moved or the mouse button is released.) When `TrackMouseLocation` returns, the `outResult` parameter contains a value representing the type of mouse activity that

occurred (press, release, and so on) and the `outPt` parameter contains the mouse location.

The function `TrackMouseRegion` is similar to `TrackMouseLocation` except that `TrackMouseRegion` only returns when the mouse enters or exits a specified region.

Alternative for Delay Function on Mac OS X

Programs sometimes call the function `Delay` to pause program execution for the number of ticks passed in the `duration` parameter. On Mac OS X, if the delay is more than about two seconds, the cursor will automatically be set to the wait cursor. To avoid this, you can instead call the function `RunCurrentEventLoop` with the required delay in seconds (perhaps converted from ticks using the macro `TicksToEventTime`) passed in the `inTimeout` parameter.

Main Constants, Data Types, and Functions

Constants

Error Codes

```
eventAlreadyPostedErr           = -9860
eventClassInvalidErr            = -9862
eventClassIncorrectErr          = -9864
eventHandlerAlreadyInstalledErr = -9866
eventInternalErr                = -9868
eventKindIncorrectErr           = -9869
eventParameterNotFoundErr       = -9870
eventNotHandledErr              = -9874
eventLoopTimedOutErr            = -9875
eventLoopQuitErr                = -9876
eventNotInQueueErr              = -9877
```

Event Classes

```
kEventClassMouse                = FOUR_CHAR_CODE('mous')
kEventClassKeyboard             = FOUR_CHAR_CODE('keyb')
kEventClassTextInput            = FOUR_CHAR_CODE('text')
kEventClassApplication          = FOUR_CHAR_CODE('appl')
kEventClassMenu                 = FOUR_CHAR_CODE('menu')
kEventClassWindow               = FOUR_CHAR_CODE('wind')
kEventClassControl              = FOUR_CHAR_CODE('cntl')
kEventClassCommand              = FOUR_CHAR_CODE('cmds')
```

Event Types

```
kEventMouseDown                 = 1
kEventMouseUp                   = 2
kEventMouseMoved                = 5
kEventMouseDragged              = 6

kEventRawKeyDown                = 1
kEventRawKeyRepeat              = 2
kEventRawKeyUp                  = 3
kEventRawKeyModifiersChanged    = 4

kEventAppActivated              = 1
kEventAppDeactivated            = 2
kEventAppQuit                   = 3
kEventAppLaunchNotification     = 4
kEventMenuEnableItems           = 8

kEventWindowUpdate              = 1
kEventWindowDrawContent         = 2
kEventWindowActivated           = 5
kEventWindowDeactivated         = 6
kEventWindowGetClickActivation  = 7
kEventWindowShown               = 24
kEventWindowHidden              = 25
kEventWindowBoundsChanging      = 26
kEventWindowBoundsChanged       = 27
kEventWindowClickDragRgn        = 32
kEventWindowClickResizeRgn      = 33
kEventWindowClickCollapseRgn    = 34
```

```
kEventWindowClickCloseRgn       = 35
kEventWindowClickZoomRgn        = 36
kEventWindowClickContentRgn     = 37
kEventWindowClickProxyIconRgn   = 38

kEventControlHit

kEventProcessCommand            = 1
```

HI Commands

```
kHICommandOK                    = FOUR_CHAR_CODE('ok  ')
kHICommandQuit                  = FOUR_CHAR_CODE('quit')
kHICommandCancel                = FOUR_CHAR_CODE('not!')
kHICommandUndo                  = FOUR_CHAR_CODE('undo')
kHICommandRedo                  = FOUR_CHAR_CODE('redo')
kHICommandCut                   = FOUR_CHAR_CODE('cut ')
kHICommandCopy                  = FOUR_CHAR_CODE('copy')
kHICommandPaste                 = FOUR_CHAR_CODE('past')
kHICommandClear                 = FOUR_CHAR_CODE('clea')
kHICommandSelectAll             = FOUR_CHAR_CODE('sall')
kHICommandHide                  = FOUR_CHAR_CODE('hide')
kHICommandPreferences           = FOUR_CHAR_CODE('pref')
kHICommandZoomWindow            = FOUR_CHAR_CODE('zoom')
kHICommandMinimizeWindow        = FOUR_CHAR_CODE('mini')
kHICommandArrangeInFront        = FOUR_CHAR_CODE('frnt')
```

Mouse Tracking Result

```
kMouseTrackingMousePressed      = 1
kMouseTrackingMouseReleased     = 2
kMouseTrackingMouseExited       = 3
kMouseTrackingMouseEntered      = 4
kMouseTrackingMouseMoved        = 5
```

Data Types

```
typedef struct OpaqueEventRef *EventRef;
typedef struct OpaqueEventHandlerRef *EventHandlerRef;
typedef struct OpaqueEventHandlerCallRef *EventHandlerCallRef;
typedef struct OpaqueEventLoopRef *EventLoopRef;
typedef double EventTime;
typedef UInt16 MouseTrackingResult;
```

EventTypeSpec

```
struct EventTypeSpec
{
  UInt32 eventClass;
  UInt32 eventKind;
} ;
typedef struct EventTypeSpec EventTypeSpec;
```

HICommand

```
struct HICommand
{
  UInt32 attributes;
  UInt32 commandID;
  struct
  {
    MenuRef menuRef;
```

```
    UInt16  menuItemIndex;
  }  menu;
} ;
typedef struct HICommand HICommand;
```

Functions and Macros

Installing and Removing Event Handlers

```
OSStatus   InstallStandardEventHandler(EventTargetRef inTarget);
OSStatus   InstallEventHandler(EventTargetRef inTarget,EventHandlerUPP inHandler,
                           UInt32 inNumTypes,const EventTypeSpec *inList,
                           void *inUserData,EventHandlerRef *outRef);
#define    InstallApplicationEventHandler(h,n,l,u,r) \
               InstallEventHandler(GetApplicationEventTarget(),(h),(n),(l),(u),(r))
#define    InstallWindowEventHandler(t,h,n,l,u,r) \
               InstallEventHandler(GetWindowEventTarget(t),(h),(n),(l),(u),(r))
#define    InstallControlEventHandler(t,h,n,l,u,r) \
               InstallEventHandler(GetControlEventTarget(t),(h),(n),(l),(u),(r))
#define    InstallMenuEventHandler(t,h,n,l,u,r) \
               InstallEventHandler(GetMenuEventTarget(t),(h),(n),(l),(u),(r))
OSStatus   RemoveEventHandler(EventHandlerRef inHandlerRef);
OSStatus   AddEventTypesToHandler(EventHandlerRef inHandlerRef,UInt32 inNumTypes,
                              const EventTypeSpec *inList);
OSStatus   RemoveEventTypesFromHandler(EventHandlerRef inHandlerRef, inNumTypes,
                                   const EventTypeSpec *inList);
```

Creating and Disposing of Event Handler UPPs

```
EventHandlerUPP NewEventHandlerUPP(EventHandlerProcPtr userRoutine);
void            DisposeEventHandlerUPP(EventHandlerUPP userUPP);
```

Running and Quitting Application Event Loop

```
void       RunApplicationEventLoop(void);
void       QuitApplicationEventLoop(void);
```

Getting Event Class and Kind

```
UInt32     GetEventClass(EventRef inEvent);
UInt32     GetEventKind(EventRef inEvent);
```

Testing for User Cancelled

```
Boolean    IsUserCancelEventRef(EventRef event);
```

Getting Data From Events

```
OSStatus   GetEventParameter(EventRef inEvent,EventParamName inName,
                           EventParamType inDesiredType,EventParamType *outActualType,
                           UInt32 inBufferSize,UInt32 *outActualSize,void *ioBuffer);
```

Converting an Event Reference to an EventRecord

```
Boolean    ConvertEventRefToEventRecord(EventRef inEvent,EventRecord *outEvent);
```

Sending Events

```
OSStatus   SendEventToEventTarget(EventRef inEvent,EventTargetRef inTarget);
#define    SendEventToApplication(e) \
```

```
                SendEventToEventTarget((e),GetApplicationEventTarget())
#define     SendEventToWindow(e,t) \
                SendEventToEventTarget((e),GetWindowEventTarget(t))
#define     SendEventToControl(e,t) \
                SendEventToEventTarget((e),GetControlEventTarget(t))
#define     SendEventToMenu(e,t) \
                SendEventToEventTarget((e),GetMenuEventTarget(t))
#define     SendEventToUserFocus(e) \
                SendEventToEventTarget((e),GetUserFocusEventTarget())
```

Installing, Resetting, and Removing Timers

```
OSStatus    InstallEventLoopTimer(EventLoopRef inEventLoop, EventTimerInterval inFireDelay,
                                  EventTimerInterval inInterval,
                                  EventLoopTimerUPP inTimerProc,void *inTimerData,
                                  EventLoopTimerRef *outTimer);
OSStatus    SetEventLoopTimerNextFireTime(EventLoopTimerRef inTimer,
                                          EventTimerInterval inNextFire);
OSStatus    RemoveEventLoopTimer(EventLoopTimerRef inTimer);
```

Calling Through to Handlers Below Current Handler

```
OSStatus    CallNextEventHandler(EventHandlerCallRef inCallRef,EventRef inEvent);
```

Getting Event and System Time

```
EventTime   GetEventTime(EventRef inEvent);
EventTime   GetCurrentEventTime(void);
```

Converting Between Ticks and EventTime

```
#define TicksToEventTime(t)  (EventTime) ((t) / 60.0)
#define EventTimeToTicks(t)  (UInt32)    ((t) * 60)
```

Mouse Tracking

```
OSStatus    TrackMouseLocation(GrafPtr inPort,Point *outPt,MouseTrackingResult *outResult);
OSStatus    TrackMouseRegion(GrafPtr inPort,RgnHandle inRegion,Boolean *ioWasInRgn,
            MouseTrackingResult *outResult);
```

Relevant Window Manager Constants and Functions

Constants

```
kWindowStandardHandlerAttribute = (1L << 25)
```

Functions

```
OSStatus   ChangeWindowAttributes(WindowRef window,WindowAttributes setTheseAttributes,
                                  WindowAttributes clearTheseAttributes);
```

Demonstration Program CarbonEvents1 Listing

```
//
**********************************************************************************************
// CarbonEvents1.c                                                          CARBON EVENT
MODEL
//
**********************************************************************************************
//
// This program opens a kWindowFullZoomGrowDocumentProc window, creates a root control for
// the window (on Mac OS 8/9), and adds a pop-up menu button control to the window.
//
// The standard application event handler handles all application events. The standard
// window event handler is installed on the window. In addition, the program installs its
own
// handler on the window for the purpose of determining which item the user chooses in the
// pop-up menu button's menu. (Although installed on the window, this handler could just as
// easily be installed on the control.)
//
// The program utilizes the following resources:
//
// •  A 'plst' resource.
//
// •  An 'MBAR' resource, and 'MENU' resources for OS9Apple/Application, File, and Edit
menus
//    and the pop-up menu (preload, non-purgeable).
//
// •  A 'WIND' resource (purgeable) (initially not visible).
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
//
**********************************************************************************************

// ..........................................................................
includes

#include <Carbon.h>

// ..........................................................................
defines
#define rMenubar   128
#define rWindow    128
#define mFile      129
#define  iQuit     12
#define mPopupMenu 131
// ..................................................................... global variables

Boolean gRunningOnX = false;

// ..................................................................... function prototypes

void      main               (void);
void      doPreliminaries    (void);
OSStatus  windowEventHandler (EventHandlerCallRef,EventRef,void *);
void      doNewWindow        (void);
void      doGetControls      (WindowRef);

// ************************************************************************************
main
```

```
void  main(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;

  // ..................................................................... do
preliminaries

  doPreliminaries();

  // ..................................................................... set up menu bar and
menus

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    ExitToShell();
  SetMenuBar(menubarHdl);
  DrawMenuBar();

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
    }

    gRunningOnX = true;
  }
  else
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef !        = NULL)
      SetMenuItemCommandID(menuRef,iQuit,kHICommandQuit);
  }

  // ..................................................................... open
window

  doNewWindow();
  // ..................................................... run application event
loop

  RunApplicationEventLoop();
}

// ************************************************************************
doPreliminaries

void  doPreliminaries(void)
{
  MoreMasterPointers(32);
  InitCursor();
}

// ************************************************************************
doNewWindow

void  doNewWindow(void)
{
  WindowRef     windowRef;
```

```
  OSStatus      osError;
  Rect          controlRect = {  42,39,62,235 } ;
  ControlRef    controlRect;
  EventTypeSpec windowEvents[] = {  {  kEventClassControl, kEventControlHit }  } ;

  // ..................................................... open window and set
attributes

  if(!(windowRef = GetNewCWindow(rWindow,NULL,(WindowRef) -1)))
    ExitToShell();
  SetPortWindowPort(windowRef);

  ChangeWindowAttributes(windowRef,kWindowStandardHandlerAttribute ,0);

  // ..................................................... install window event
handler

  InstallWindowEventHandler(windowRef,
                            NewEventHandlerUPP((EventHandlerProcPtr) windowEventHandler),
                            GetEventTypeCount(windowEvents),windowEvents,0,NULL);

  // ........................... create root control for window and get popup button
control

  if(!gRunningOnX)
    CreateRootControl(windowRef,&controlRef);

  if((osError = CreatePopupButtonControl(windowRef,&controlRect,CFSTR("Time Zone:"),
                                         mPopupMenu,false,-1,0,0,&controlRef)) != noErr)
    ExitToShell();

  // ..................................................................... show
window

  ShowWindow(windowRef);
}
// ************************************************************************
windowEventHandler

OSStatus  windowEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                             void* userData)
{
  OSStatus    result = eventNotHandledErr;
  UInt32      eventKind;
  ControlRef  controlRef;
  MenuRef     menuRef;
  Size        actualSize;
  SInt16      controlValue;
  Str255      menuItemString;
  Rect        theRect = {  0,0,40,293 } ;
  CFStringRef stringRef;
  Rect        textBoxRect;

  eventKind  = GetEventKind(eventRef);

  if(eventKind == kEventControlHit)
  {
    GetEventParameter(eventRef,kEventParamDirectObject,typeControlRef,NULL,
                      sizeof(controlRef),NULL,&controlRef);

    GetControlData(controlRef,kControlEntireControl,kControlPopupButtonMenuHandleTag,
                   sizeof(menuRef),&menuRef,&actualSize);
    controlValue = GetControlValue(controlRef);
    GetMenuItemText(menuRef,controlValue,menuItemString);
```

```
    EraseRect(&theRect);
    stringRef = CFStringCreateWithPascalString(NULL,menuItemString,
                                               kCFStringEncodingMacRoman);
    SetRect(&textBoxRect,theRect.left,7,theRect.right,22);
    DrawThemeTextBox(stringRef,kThemeSmallSystemFont,true,true,&textBoxRect,teJustCenter,
                     NULL);
    if(stringRef != NULL)
      CFRelease(stringRef);

    result = noErr;
  }

  return result;
}

//
*****************************************************************************************
```

Demonstration Program CarbonEvents1 Comments

When this program is run, the user should

- Drag, resize, zoom and, when done, close the window.
- Send the application to the background and bring it to the foreground, noting the activation and deactivation of the pop-up menu button control.
- Choose items in the pop-up menu button's menu.
- Quit the application by choosing the Quit item in the Mac OS 9 File/Mac OS X Application menu and using its Command-key equivalent.

main

If the program is running on OS 8/9, `SetMenuItemCommandID` is called to assign the command ID 'quit' to the Quit item in the File menu. (This command is assigned to the Mac OS X Quit item by default.) Thus, when the Quit item is chosen on Mac OS 8/9 and Mac OS X, the standard application event handler will call the default Quit Application Apple event handler (automatically installed when `RunApplicationEventLoop` is called) to close down the program.

The standard application event handler is installed when `RunApplicationEventLoop` is called. The standard application event handler handles all application events, including, in Classic event model parlance, suspend and resume events (that is, application-deactivated and application-activated events).

doNewWindow

After the window is created, `ChangeWindowAttributes` is called to set the standard handler attribute, causing the standard window event handler to be installed on the window. The standard window event handles all window dragging, sizing, zooming, collapsing/minimizing, and closing operations, attends to control updating, and (provided a root control is created for the window), control deactivation and activation when the program is sent to the back and brought to the. It also calls `TrackControl` when a mouse-down occurs in the pop-up menu button, thus handling all user interaction with the control.

The call to `InstallWindowEventHandler` installs the application's window event handler on the window. A single event type (`kEventControlHit`) is registered. Note

that this handler could have been installed on the control itself, but is installed on the window in this program for the purpose of emphasizing the propagation order of events. (No handler is installed on the control, so the event will "fall through" to the window event handler.)

If the program is running on Mac OS 8/9, `CreateRootControl` is called to create a root control for the window. (This call is not necessary on Mac OS X because, on Mac OS X, a root control is automatically created on windows that have at least one control.)

windowEventHandler

`windowEventHandler` is a callback function. It is the window event handler installed on the window by the call to `InstallWindowEventHandler` in main. Its purpose is to determine the control value of the pop-up menu button control, and thus the menu item the user chose.

As previously stated, the standard window event handler calls `TrackControl` when a mouse-down occurs in the pop-up menu button. The Carbon Event Manager sends the `kEventControlHit` event type when `TrackControl` returns (regardless, incidentally, of whether the cursor is still within the control when the mouse button is released).

`GetEventType` is called to get the event type. If the event type is `kEventControlHit`, the if block executes and the handler returns `noErr`, indicating to the Carbon Event Manager that the event has been fully handled and that it should not be propagated further. If the event type is not `kEventControlHit`, the handler returns `eventNotHandledErr`, indicating that the event should be propagated further.

Within the if block, `GetEventParameter` is called to extract certain data from the event, specifically, a reference to the control. This reference is passed in the call to `GetControlData`, which gets a reference to the control's menu. The call to `GetControlValue` then gets the control's value, and the call to `GetMenuItemText` gets the text of the menu item. This text is then drawn at the top of the window to prove that the handler has done its job.

Demonstration Program CarbonEvents2 Listing

```
//
****************************************************************************************
// CarbonEvents2.h                                                    CARBON EVENT
MODEL
//
****************************************************************************************
//
// This program allows the user to:
//
// •  Open, close, and interact with kWindowFullZoomGrowDocumentProc windows containing
//    various controls.
//
// •  Open, close and interact with a movable modal dialog and, on Mac OS X only,
//    window-modal (sheet) alerts and window-modal (sheet) dialogs.
//
// The program demonstrates the main aspects of the Carbon Event model, specifically:
//
// •  Events relating to menus, windows and controls, including the detection of mouse-downs
//    in controls in document windows and movable modal dialogs.
//
// •  Keyboard events.
//
// •  Events relating to application activation and deactivation (resume and suspend in
//    Classic event model parlance).
//
// •  The use of mouse-moved events in support of cursor adjustment functions.
//
// •  The installation of event loop timers (used, in this program, to trigger an "idle"
//    function.
//
// The program also demonstrates the implementation of live window resizing.
//
// The window contains a window header frame in which is displayed the menu items chosen
from
// pop-up menu buttons, the identity of a push button when that push button is clicked, and
// scroll bar control values when the scroll arrows or gray areas/track of the scroll bars
are
// clicked and when the scroll box/scroller is dragged. (The vertical scroll bar is the
// non-live feedback variant. The horizontal scroll bar is the live-feedback variant.)  Text
// extracted from the edit text item in the window-modal (sheet) dialog and the identity of
// the button clicked in the window-modal (sheet) alert are also displayed in the window
// header frame.
//
// The movable modal dialog serves the secondary purpose of proving window correct window
// updating even though an event filter function is not used by the dialog.
//
// The program utilizes the following resources:
//
// •  A 'plst' resource.
//
// •  An 'MBAR' resource, and 'MENU' resources for OS9Apple/Application, File, Edit, and
//    Typing Target, and Dialogs menus, and the pop-up menus (preload, non-purgeable).
//
// •  A 'WIND' resource (purgeable) (initially not visible).
//
// •  A 'DLOG' resource ((purgeable) (initially not visible), with associated 'DITL', 'dlgx'
//    and 'dfnt' resources, for the window-modal (sheet) dialog.
//
// •  A 'CNTL' resource (purgeable) for an image well control in the window-modal (sheet)
//    dialog.
```

```
//
// •  A 'STR#' resource (purgeable) containing text for the window-modal (sheet) alert.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
//
*****************************************************************************

// ...........................................................................
includes

#include <Carbon.h>

// ...........................................................................
defines

#define rMenubar                128
#define rWindow                 128
#define rAboutDialog            128
#define mAppleApplication       128
#define  iAbout                 1
#define mFile                   129
#define  iQuit                  12
#define  iNew                   1
#define  iClose                 4
#define mTyping                 131
#define  iDocument              1
#define  iEditTextControl       2
#define  iAllOfTheAbove         3
#define mDialogs                132
#define  iMovableModal          1
#define  iSheetAlert            2
#define  iSheetDialog           3
#define mWindow                 135
#define rSheetDialog            128
#define rSheetStrings           128
#define  sAlertSheetMessage     1
#define  sAlertSheetInformative 2
#define kPopupCountryID         'ctry'
#define kScrollBarWidth         15
#define MIN(a,b)                ((a) < (b) ? (a) : (b))
#define topLeft(r)              (((Point *) &(r))[0])
#define botRight(r)             (((Point *) &(r))[1])

// ...........................................................................
typedefs

typedef struct
{
  ControlRef popupTimeZoneRef;
  ControlRef popupCountryRef;
  ControlRef radiobuttonRedRef;
  ControlRef radiobuttonWhiteRef;
  ControlRef radiobuttonBlueRef;
  ControlRef groupboxColorRef;
  ControlRef groupboxTypingRef;
  ControlRef buttonRef;
  ControlRef buttonDefaultRef;
  ControlRef editTextRef;
  ControlRef scrollbarVertRef;
  ControlRef scrollbarHorizRef;
}  docStruc, **docStrucHandle;

// ..................................................................... function
prototypes
```

```
void            main                  (void);
void            doPreliminaries       (void);
OSStatus        appEventHandler       (EventHandlerCallRef,EventRef,void *);
OSStatus        windowEventHandler    (EventHandlerCallRef,EventRef,void *);
void            doNewWindow           (void);
EventHandlerUPP doGetHandlerUPP       (void);
void            doCloseWindow         (WindowRef);
void            doGetControls         (WindowRef);
void            doIdle                (void);
void            doAdjustMenus         (void);
void            doMenuChoice          (MenuID,MenuItemIndex);
void            doDrawContent         (WindowRef);
void            doActivateDeactivate  (WindowRef,Boolean);
void            doControlHit1         (WindowRef,ControlRef,Point,ControlPartCode);
void            doControlHit2         (void);
void            doPopupMenuChoice     (WindowRef,ControlRef,SInt16);
void            doVertScrollbar       (ControlPartCode,WindowRef,ControlRef,Point);
void            actionFunctionVert    (ControlRef,ControlPartCode);
void            actionFunctionHoriz   (ControlRef,ControlPartCode);
void            doMoveScrollBox       (ControlRef,SInt16);
void            doRadioButtons        (ControlRef,WindowRef);
void            doCheckboxes          (ControlRef);
void            doPushButtons         (ControlRef,WindowRef);
void            doAdjustScrollBars    (WindowRef);
void            doAdjustCursor        (WindowRef);
void            doDrawDocumentTyping  (SInt8,UInt32);
void            doDrawMessage         (WindowRef,Boolean);
void            doConcatPStrings      (Str255,Str255);
void            doCopyPString         (Str255,Str255);
OSStatus        doSheetAlert          (void);
OSStatus        doSheetDialog         (void);
EventHandlerUPP doGetSheetHandlerUPP  (void);
OSStatus        sheetEventHandler     (EventHandlerCallRef,EventRef,void *);
OSStatus        doMovableModalDialog  (void);
EventHandlerUPP doGetDialogHandlerUPP (void);
OSStatus        dialogEventHandler    (EventHandlerCallRef,EventRef,void *);

//
**********************************************************************************************
// CarbonEvents2.c
//
**********************************************************************************************

// ..........................................................................
includes

#include "CarbonEvents2.h"

// ..................................................................... global
variables

ControlActionUPP gActionFunctionVertUPP;
ControlActionUPP gActionFunctionHorizUPP;
Boolean          gRunningOnX = false;
SInt16           gNumberOfWindows = 0;
Str255           gCurrentString;
SInt16           gTypingTarget = 3;

// **********************************************************************************
main

void  main(void)
{
  MenuBarHandle     menubarHdl;
  SInt32            response;
```

```
  MenuRef           menuRef;
  EventLoopTimerUPP eventLoopTimerUPP;
  EventTypeSpec   applicationEvents[] = {  {  kEventClassApplication,
kEventAppActivated    } ,
                                          {  kEventClassCommand,
kEventProcessCommand  } ,
                                          {  kEventClassMenu,
kEventMenuEnableItems } ,
                                          {  kEventClassMouse,
kEventMouseMoved      }  } ;

  // ..................................................................... do
preliminaries

  doPreliminaries();

  // ..................................................create universal procedure
pointers

  gActionFunctionVertUPP  = NewControlActionUPP((ControlActionProcPtr) actionFunctionVert);
  gActionFunctionHorizUPP = NewControlActionUPP((ControlActionProcPtr) actionFunctionHoriz);

  // ............................................................ set up menu bar and
menus

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    ExitToShell();
  SetMenuBar(menubarHdl);
  CreateStandardWindowMenu(0,&menuRef);
  SetMenuID(menuRef,mWindow);
  InsertMenu(menuRef,0);

  DrawMenuBar();

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
    }

    gRunningOnX = true;
  }
  else
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
      SetMenuItemCommandID(menuRef,iQuit,kHICommandQuit);

    menuRef = GetMenuRef(mDialogs);
    if(menuRef != NULL)
    {
      DisableMenuItem(menuRef,iSheetAlert);
      DisableMenuItem(menuRef,iSheetDialog);
    }
  }

  // ..................................... initial advisory text for window header

  doCopyPString("\pManipulate the window and controls. Do typing.",gCurrentString);

  //     ..................................... install application event handler
```

```
  InstallApplicationEventHandler(NewEventHandlerUPP((EventHandlerProcPtr) appEventHandler),
                                 GetEventTypeCount(applicationEvents),applicationEvents,
                                 0,NULL);

  // ..................................................................... install
timer

  eventLoopTimerUPP = NewEventLoopTimerUPP((EventLoopTimerProcPtr) doIdle);

  InstallEventLoopTimer(GetCurrentEventLoop(),0,TicksToEventTime(GetCaretTime()),
                        eventLoopTimerUPP,NULL,NULL);

  // ..................................................................... open
window

  doNewWindow();
  // ..................................................... run application event
loop

  RunApplicationEventLoop();
}

// ************************************************************************
doPreliminaries

void  doPreliminaries(void)
{
  MoreMasterPointers(128);
  InitCursor();
}

// ************************************************************************
appEventHandler

OSStatus  appEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                          void * userData)
{
  OSStatus      result = eventNotHandledErr;
  UInt32        eventClass;
  UInt32        eventKind;
  HICommand     hiCommand;
  MenuID        menuID;
  MenuItemIndex menuItem;
  WindowClass   windowClass;

  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);

  switch(eventClass)
  {
    case kEventClassApplication:
      if(eventKind == kEventAppActivated)
        SetThemeCursor(kThemeArrowCursor);
      break;

    case kEventClassCommand:
      if(eventKind == kEventProcessCommand)
      {
        GetEventParameter(eventRef,kEventParamDirectObject,typeHICommand,NULL,
                          sizeof(HICommand),NULL,&hiCommand);
        if(hiCommand.commandID == kHICommandQuit)
          result = eventNotHandledErr;
        menuID = GetMenuID(hiCommand.menu.menuRef);
        menuItem = hiCommand.menu.menuItemIndex;
        if((hiCommand.commandID != kHICommandQuit) &&
```

```
           (menuID >= mAppleApplication && menuID <= mDialogs))
        {
          doMenuChoice(menuID,menuItem);
          result = noErr;
        }
        if(hiCommand.commandID == kPopupCountryID)
        {
          doControlHit2();
          result = noErr;
        }
      }
      break;

    case kEventClassMenu:
      if(eventKind == kEventMenuEnableItems)
      {
        GetWindowClass(FrontWindow(),&windowClass);
        if(windowClass == kDocumentWindowClass)
          doAdjustMenus();
        result = noErr;
      }
      break;

    case kEventClassMouse:
      if(eventKind == kEventMouseMoved)
      {
        GetWindowClass(FrontWindow(),&windowClass);
        if(windowClass == kDocumentWindowClass)
          doAdjustCursor(FrontWindow());
        result = noErr;
      }
      break;
  }

  return result;
}

// ************************************************************************
windowEventHandler

OSStatus  windowEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                             void* userData)
{
  OSStatus        result = eventNotHandledErr;
  UInt32          eventClass;
  UInt32          eventKind;
  WindowRef       windowRef;
  Rect            mainScreenRect, portRect;
  BitMap          screenBits;
  Point           idealHeightAndWidth, minimumHeightAndWidth, mouseLocation;
  ControlRef      controlRef;
  ControlPartCode controlPartCode;
  SInt8           charCode;
  UInt32          modifiers;
  HICommand       hiCommand;

  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);

  switch(eventClass)
  {
    case kEventClassWindow:                                          // event class
window
      GetEventParameter(eventRef,kEventParamDirectObject,typeWindowRef,NULL,sizeof
(windowRef),
                        NULL,&windowRef);
```

```
      switch(eventKind)
      {
        case kEventWindowDrawContent:
          doDrawContent(windowRef);
          break;

        case kEventWindowActivated:
          doActivateDeactivate(windowRef,true);
          break;

        case kEventWindowDeactivated:
          doActivateDeactivate(windowRef,false);
          break;

        case kEventWindowGetIdealSize:
          mainScreenRect = GetQDGlobalsScreenBits(&screenBits)->bounds;
          idealHeightAndWidth.v = mainScreenRect.bottom - 75;
          idealHeightAndWidth.h = 600;
          SetEventParameter(eventRef,kEventParamDimensions,typeQDPoint,
                            sizeof(idealHeightAndWidth),&idealHeightAndWidth);
           result = noErr;
          break;

        case kEventWindowGetMinimumSize:
          minimumHeightAndWidth.v = 308;
          minimumHeightAndWidth.h = 290;
          SetEventParameter(eventRef,kEventParamDimensions,typeQDPoint,
                            sizeof(minimumHeightAndWidth),&minimumHeightAndWidth);
          result = noErr;
          break;

        case kEventWindowZoomed:
          GetWindowPortBounds(windowRef,&portRect);
          EraseRect(&portRect);
          doAdjustScrollBars(windowRef);
          result = noErr;
          break;

        case kEventWindowBoundsChanged:
          doAdjustScrollBars(windowRef);
          doDrawMessage(windowRef,true);
          result = noErr;
          break;

        case kEventWindowClose:
          doCloseWindow(windowRef);
          break;
      }
      break;
    case kEventClassControl:                                        // event class
control
      switch(eventKind)
      {
        case kEventControlClick:
          GetEventParameter(eventRef,kEventParamMouseLocation,typeQDPoint,NULL,
                            sizeof(mouseLocation),NULL,&mouseLocation);
          SetPortWindowPort(FrontWindow());
          GlobalToLocal(&mouseLocation);
          controlRef = FindControlUnderMouse(mouseLocation,FrontWindow(),&controlPartCode);
          if(controlRef)
          {
            doControlHit1(FrontWindow(),controlRef,mouseLocation,controlPartCode);
            result = noErr;
          }
          break;
```

```
      }
      break;

    case kEventClassKeyboard:                                          // event class
keyboard
      switch(eventKind)
      {
        case kEventRawKeyDown:
          if(gTypingTarget == 1 || gTypingTarget == 3)
          {
            GetEventParameter(eventRef,kEventParamKeyMacCharCodes,typeChar,NULL,
                              sizeof(charCode),NULL,&charCode);
            GetEventParameter(eventRef,kEventParamKeyModifiers,typeUInt32,NULL,
                              sizeof(modifiers),NULL,&modifiers);
            doDrawDocumentTyping(charCode,modifiers);
          }
          if(gTypingTarget == 1)
            result = noErr;
          break;
      }
      break;

    case kEventClassCommand:                                            // event class
command
      if(eventKind == kEventProcessCommand)
      {
        GetEventParameter(eventRef,kEventParamDirectObject,typeHICommand,NULL,
                          sizeof(HICommand),NULL,&hiCommand);
        if(hiCommand.commandID == kHICommandOK)
          doCopyPString("\pOK button hit",gCurrentString);
        if(hiCommand.commandID == kHICommandCancel)
          doCopyPString("\pCancel button hit",gCurrentString);
        if(hiCommand.commandID == kHICommandOther)
          doCopyPString("\pOther button hit",gCurrentString);
        GetWindowPortBounds(FrontWindow(),&portRect);
        InvalWindowRect(FrontWindow(),&portRect);
      }
      break;
  }

  return result;
}
// ************************************************************************
doNewWindow

void  doNewWindow(void)
{
  WindowRef      windowRef;
  Str255         windowTitleString = "\pCarbonEvents2 - ";
  Str255         theString;
  docStrucHandle docStrucHdl;
  SInt16         a;
  MenuRef        menuRef;
  EventTypeSpec  windowEvents[] = {  {  kEventClassWindow,   kEventWindowDrawContent    } ,
                                     {  kEventClassWindow,   kEventWindowActivated      } ,
                                     {  kEventClassWindow,   kEventWindowDeactivated    } ,
                                     {  kEventClassWindow,   kEventWindowGetIdealSize   } ,
                                     {  kEventClassWindow,   kEventWindowGetMinimumSize } ,
                                     {  kEventClassWindow,   kEventWindowZoomed         } ,
                                     {  kEventClassWindow,   kEventWindowBoundsChanged  } ,
                                     {  kEventClassWindow,   kEventWindowClose          } ,
                                     {  kEventClassControl,  kEventControlClick         } ,
                                     {  kEventClassKeyboard, kEventRawKeyDown           } ,
                                     {  kEventClassCommand,
kEventProcessCommand       }  } ;
```

```
  // ..................................................... open window and set
attributes

  if(!(windowRef = GetNewCWindow(rWindow,NULL,(WindowRef) -1)))
    ExitToShell();

  ChangeWindowAttributes(windowRef,kWindowStandardHandlerAttribute,0);
  if(gRunningOnX)
    ChangeWindowAttributes(windowRef,kWindowLiveResizeAttribute,0);

  // ..................................... alternative open window and set
attributes
  // Rect              contentRect = {  100,100,405,393 } ;
  // WindowAttributes  attributes  = kWindowStandardHandlerAttribute |
  //                                 kWindowStandardDocumentAttributes |
  //                                 kWindowLiveResizeAttribute;
  //
  // CreateNewWindow(kDocumentWindowClass,attributes,&contentRect,&windowRef);
  // RepositionWindow(windowRef,NULL,kWindowAlertPositionOnMainScreen);

  // ........... get block for document structure, assign handle to window record refCon
field

  if(!(docStrucHdl = (docStrucHandle) NewHandle(sizeof(docStruc))))
    ExitToShell();

  // ..................................................................... set window
title

  SetWRefCon(windowRef,(SInt32) docStrucHdl);
  gNumberOfWindows ++;
  NumToString(gNumberOfWindows,theString);
  doConcatPStrings(windowTitleString,theString);
  SetWTitle(windowRef,windowTitleString);

  SetPortWindowPort(windowRef);
  TextSize(46);
  // ........................... if running on Mac OS 8/9, set theme-compliant color/
pattern

  SetThemeWindowBackground(windowRef,kThemeBrushDialogBackgroundActive,false);

  //..................................................... install window event
handler

  InstallWindowEventHandler(windowRef,doGetHandlerUPP(),GetEventTypeCount(windowEvents),
                            windowEvents,0,NULL);

  // ..................................... get controls, adjust scroll bars, and show
window

  doGetCo       ntrols(windowRef);
  doAdjustScrollBars(windowRef);
  ShowWindow(windowRef);

  //..................... enable Typing and Window menu, fix typing target and keyboard
focus

  menuRef = GetMenuRef(mTyping);
  EnableMenuItem(menuRef,0);

  for(a = iDocument;a <= iAllOfTheAbove;a ++)
    CheckMenuItem(menuRef,a,false);
  CheckMenuItem(menuRef,iAllOfTheAbove,true);
  SetKeyboardFocus(windowRef,(*docStrucHdl)->editTextRef,kControlFocusNextPart);
  gTypingTarget = 3;
```

```
  EnableMenuItem(GetMenuRef(mWindow),0);
}

// ************************************************************************
doGetHandlerUPP

EventHandlerUPP  doGetHandlerUPP(void)
{
  static EventHandlerUPP windowEventHandlerUPP;

  if(windowEventHandlerUPP == NULL)
    windowEventHandlerUPP = NewEventHandlerUPP((EventHandlerProcPtr) windowEventHandler);

  return windowEventHandlerUPP;
}

// ************************************************************************
doCloseWindow

void  doCloseWindow(WindowRef windowRef)
{
  docStrucHandle docStrucHdl;

  KillControls(windowRef);

  docStrucHdl = (docStrucHandle) (GetWRefCon(windowRef));
  DisposeHandle((Handle) docStrucHdl);

  gNumberOfWindows --;
  if(gNumberOfWindows == 0)
  {
    DisableMenuItem(GetMenuRef(mTyping),0);
    DisableMenuItem(GetMenuRef(mWindow),0);
  }
}

// ************************************************************************
doGetControls

void  doGetControls(WindowRef windowRef)
{
  ControlRef     controlRef;
  docStrucHandle docStrucHdl;
  OSStatus       osError;
  Rect           controlRect;
  Boolean        booleanData = true;

  CreateRootControl(windowRef,&controlRef);

  docStrucHdl = (docStrucHandle) (GetWRefCon(windowRef));

  SetRect(&controlRect,40,40,235,60);
  if((osError = CreatePopupButtonControl(windowRef,&controlRect,CFSTR("Time Zone:"),133,
false,
                                       -1,0,0,&(*docStrucHdl)->popupTimeZoneRef)) !=
noErr)
    ExitToShell();

  SetRect(&controlRect,55,73,235,93);
  if((osError = CreatePopupButtonControl(windowRef,&controlRect,CFSTR("Country:"),134,false,
                                       -1,0,0,&(*docStrucHdl)->popupCountryRef)) != noErr)
    ExitToShell();

  SetRect(&controlRect,35,126,91,144);
  if((osError = CreateRadioButtonControl(windowRef,&controlRect,CFSTR("Red"),1,false,
```

```
                                        &(*docStrucHdl)->radiobuttonRedRef)) != noErr)
    ExitToShell();

  SetRect(&controlRect,35,148,91,166);
  if((osError = CreateRadioButtonControl(windowRef,&controlRect,CFSTR("White"),0,false,
                                        &(*docStrucHdl)->radiobuttonWhiteRef)) != noErr)
    ExitToShell();

  SetRect(&controlRect,35,170,91,188);
  if((osError = CreateRadioButtonControl(windowRef,&controlRect,CFSTR("Blue"),0,false,
                                        &(*docStrucHdl)->radiobuttonBlueRef)) != noErr)
    ExitToShell();

  SetRect(&controlRect,20,102,107,201);
  if((osError = CreateGroupBoxControl(windowRef,&controlRect,CFSTR("Color"),true,
                                      &(*docStrucHdl)->groupboxColorRef)) != noErr)
    ExitToShell();

  SetRect(&controlRect,123,102,255,201);
  if((osError = CreateGroupBoxControl(windowRef,&controlRect,CFSTR("Typing"),true,
                                      &(*docStrucHdl)->groupboxTypingRef)) != noErr)
    ExitToShell();
  SetRect(&controlRect,63,213,132,233);
  if((osError = CreatePushButtonControl(windowRef,&controlRect,CFSTR("Cancel"),
                                       &(*docStrucHdl)->buttonRef)) != noErr)
    ExitToShell();

  SetRect(&controlRect,144,213,213,233);
  if((osError = CreatePushButtonControl(windowRef,&controlRect,CFSTR("OK"),
                                       &(*docStrucHdl)->buttonDefaultRef)) != noErr)
    ExitToShell();

  SetRect(&controlRect,26,251,249,267);
  if((osError = CreateEditTextControl(windowRef,&controlRect,NULL,false,true,NULL,
                                      &(*docStrucHdl)->editTextRef)) != noErr)
    ExitToShell();

  SetRect(&controlRect,0,0,16,262);
  if((osError = CreateScrollBarControl(windowRef,&controlRect,0,0,125,0,false,NULL,
                                       &(*docStrucHdl)->scrollbarVertRef)) != noErr)
    ExitToShell();

  SetRect(&controlRect,0,0,245,16);
  if((osError = CreateScrollBarControl(windowRef,&controlRect,0,0,125,0,true,NULL,
                                       &(*docStrucHdl)->scrollbarHorizRef)) != noErr)
    ExitToShell();

  AutoEmbedControl((*docStrucHdl)->radiobuttonRedRef,windowRef);
  AutoEmbedControl((*docStrucHdl)->radiobuttonWhiteRef,windowRef);
  AutoEmbedControl((*docStrucHdl)->radiobuttonBlueRef,windowRef);

  SetControlCommandID((*docStrucHdl)->popupCountryRef,kPopupCountryID);

  SetControlData((*docStrucHdl)->buttonDefaultRef,kControlEntireControl,
                 kControlPushButtonDefaultTag,sizeof(booleanData),&booleanData);
}

// ********************************************************************************
doIdle

void  doIdle(void)
{
  if(!gRunningOnX)
    IdleControls(FrontWindow());
}
```

```
// ****************************************************************************
doAdjustMenus

void  doAdjustMenus(void)
{
  MenuRef   menuRef;
  OSStatus  osError;
  WindowRef windowRef;
  if(FrontWindow())
  {
    menuRef = GetMenuRef(mFile);
    EnableMenuItem(menuRef,iClose);
    menuRef = GetMenuRef(mTyping);
    EnableMenuItem(menuRef,0);
  }
  else
  {
    menuRef = GetMenuRef(mFile);
    DisableMenuItem(menuRef,iClose);
    menuRef = GetMenuRef(mTyping);
    DisableMenuItem(menuRef,0);
  }

  if(gRunningOnX)
  {
    if((osError = GetSheetWindowParent(FrontWindow(),&windowRef)) == noErr)
    {
      menuRef = GetMenuRef(mFile);
      DisableMenuItem(menuRef,iClose);
      menuRef = GetMenuRef(mTyping);
      DisableMenuItem(menuRef,0);
    }
    else
    {
      if(FrontWindow())
      {
        menuRef = GetMenuRef(mTyping);
        EnableMenuItem(menuRef,0);
      }
    }

    menuRef = GetMenuRef(mDialogs);
    if(((osError = GetSheetWindowParent(FrontWindow(),&windowRef)) == noErr) ||
       (FrontWindow() == NULL) || IsWindowCollapsed(FrontWindow()))
    {
      DisableMenuItem(menuRef,iSheetAlert);
      DisableMenuItem(menuRef,iSheetDialog);
    }
    else
    {
      EnableMenuItem(menuRef,iSheetAlert);
      EnableMenuItem(menuRef,iSheetDialog);
    }
  }
}

// ****************************************************************************
doMenuChoice

void  doMenuChoice(MenuID menuID,MenuItemIndex menuItem)
{
  WindowRef      windowRef;
  SInt16         a;
  MenuRef        menuRef;
  docStrucHandle docStrucHdl;
  if(menuID == 0)
```

```
    return;

  windowRef = FrontWindow();

  switch(menuID)
  {
    case mAppleApplication:
      if(menuItem == iAbout)
        SysBeep(10);
      break;

    case mFile:
      if(menuItem == iNew)
        doNewWindow();
      else if(menuItem == iClose)
      {
        doCloseWindow(windowRef);
        DisposeWindow(windowRef);
      }
      break;

    case mTyping:
      menuRef = GetMenuRef(mTyping);
      for(a = iDocument;a <= iAllOfTheAbove;a ++)
        CheckMenuItem(menuRef,a,false);
      CheckMenuItem(menuRef,menuItem,true);
      gTypingTarget = menuItem;
      docStrucHdl = (docStrucHandle) (GetWRefCon(windowRef));
      if(menuItem == iDocument)
        SetKeyboardFocus(windowRef,(*docStrucHdl)->editTextRef,kControlFocusNoPart);
      else
        SetKeyboardFocus(windowRef,(*docStrucHdl)->editTextRef,kControlFocusNextPart);
      break;

    case mDialogs:
      if(menuItem == iMovableModal)
        if(doMovableModalDialog() != noErr)
          ExitToShell();
      if(menuItem == iSheetAlert)
        if(doSheetAlert() != noErr)
          ExitToShell();
      if(menuItem == iSheetDialog)
        if(doSheetDialog() != noErr)
          ExitToShell();
      break;
  }
}

// *****************************************************************************
doDrawContent

void  doDrawContent(WindowRef windowRef)
{
  SetPortWindowPort(windowRef);
  doDrawMessage(windowRef,windowRef == FrontWindow());
}
// *****************************************************************************
doActivateWindow

void  doActivateDeactivate(WindowRef windowRef,Boolean becomingActive)
{
  if(becomingActive)
    doDrawMessage(windowRef,becomingActive);
  else
    doDrawMessage(windowRef,becomingActive);
}
```

```
// ****************************************************************************
doControlHit1

void  doControlHit1(WindowRef windowRef,ControlRef controlRef,Point mouseLocation,
                  ControlPartCode controlPartCode)
{
  docStrucHandle docStrucHdl;
  SInt16         controlValue;

  docStrucHdl = (docStrucHandle) (GetWRefCon(windowRef));

  if(controlRef == (*docStrucHdl)->popupTimeZoneRef)
  {
    TrackControl(controlRef,mouseLocation,(ControlActionUPP) -1);
    controlValue = GetControlValue(controlRef);
    doPopupMenuChoice(windowRef,controlRef,controlValue);
  }
  else if(controlRef == (*docStrucHdl)->scrollbarVertRef)
  {
    doVertScrollbar(controlPartCode,windowRef,controlRef,mouseLocation);
  }
  else if(controlRef == (*docStrucHdl)->scrollbarHorizRef)
  {
    TrackControl(controlRef,mouseLocation,gActionFunctionHorizUPP);
  }
  else
  {
    if(TrackControl(controlRef,mouseLocation,NULL))
    {
      if(controlRef == (*docStrucHdl)->radiobuttonRedRef ||
         controlRef == (*docStrucHdl)->radiobuttonWhiteRef ||
         controlRef == (*docStrucHdl)->radiobuttonBlueRef)
      {
        doRadioButtons(controlRef,windowRef);
      }
      if(controlRef == (*docStrucHdl)->buttonRef ||
         controlRef == (*docStrucHdl)->buttonDefaultRef)
      {
        doPushButtons(controlRef,windowRef);
      }
    }
  }
}
// ****************************************************************************
doControlHit2

void  doControlHit2(void)
{
  docStrucHandle docStrucHdl;
  ControlRef     controlRef;
  SInt16         controlValue;

  docStrucHdl = (docStrucHandle) GetWRefCon(FrontWindow());
  controlRef = (*docStrucHdl)->popupCountryRef;

  controlValue = GetControlValue(controlRef);
  doPopupMenuChoice(FrontWindow(),controlRef,controlValue);
}

// ****************************************************************************
doPopupMenuChoice

void  doPopupMenuChoice(WindowRef windowRef,ControlRef controlRef,SInt16 controlValue)
{
  MenuRef menuRef;
```

```
  Size    actualSize;

  GetControlData(controlRef,kControlEntireControl,kControlPopupButtonMenuHandleTag,
                 sizeof(menuRef),&menuRef,&actualSize);
  GetMenuItemText(menuRef,controlValue,gCurrentString);
  doDrawMessage(windowRef,true);
}

// *****************************************************************************
doVertScrollbar

void  doVertScrollbar(ControlPartCode controlPartCode,WindowRef windowRef,
                      ControlRef controlRef,Point mouseXY)
{
  Str255 valueString;

  doCopyPString("\pScroll Bar Control Value: ",gCurrentString);

  switch(controlPartCode)
  {
    case kControlIndicatorPart:
      if(TrackControl(controlRef,mouseXY,NULL))
      {
        NumToString((SInt32) GetControlValue(controlRef),valueString);
        doConcatPStrings(gCurrentString,valueString);
        doDrawMessage(windowRef,true);
      }
      break;

    case kControlUpButtonPart:
    case kControlDownButtonPart:
    case kControlPageUpPart:
    case kControlPageDownPart:
      TrackControl(controlRef,mouseXY,gActionFunctionVertUPP);
      break;
  }
}
// *****************************************************************************
actionFunctionVert

void  actionFunctionVert(ControlRef controlRef,ControlPartCode controlPartCode)
{
  SInt16    scrollDistance, controlValue;
  Str255    valueString;
  WindowRef windowRef;

  doCopyPString("\pScroll Bar Control Value: ",gCurrentString);

  if(controlPartCode)
  {
    switch(controlPartCode)
    {
      case kControlUpButtonPart:
      case kControlDownButtonPart:
        scrollDistance = 2;
        break;

      case kControlPageUpPart:
      case kControlPageDownPart:
        scrollDistance = 55;
        break;
    }

    if((controlPartCode == kControlDownButtonPart) ||
       (controlPartCode == kControlPageDownPart))
      scrollDistance = -scrollDistance;
```

```
    controlValue = GetControlValue(controlRef);
    if(((controlValue == GetControlMaximum(controlRef)) && scrollDistance < 0) ||
       ((controlValue == GetControlMinimum(controlRef)) && scrollDistance > 0))
      return;

    doMoveScrollBox(controlRef,scrollDistance);

    NumToString((SInt32) GetControlValue(controlRef),valueString);
    doConcatPStrings(gCurrentString,valueString);
    windowRef = GetControlOwner(controlRef);
    doDrawMessage(windowRef,true);
  }
}

// ************************************************************************
actionFunctionHoriz

void  actionFunctionHoriz(ControlRef controlRef,ControlPartCode controlPartCode)
{
  SInt16    scrollDistance, controlValue;
  Str255    valueString;
  WindowRef windowRef;

  doCopyPString("\pScroll Bar Control Value: ",gCurrentString);
  if(controlPartCode != kControlIndicatorPart)
  {
    switch(controlPartCode)
    {
      case kControlUpButtonPart:
      case kControlDownButtonPart:
        scrollDistance = 2;
        break;

      case kControlPageUpPart:
      case kControlPageDownPart:
        scrollDistance = 55;
        break;
    }

    if((controlPartCode == kControlDownButtonPart) ||
       (controlPartCode == kControlPageDownPart))
      scrollDistance = -scrollDistance;

    controlValue = GetControlValue(controlRef);
    if(((controlValue == GetControlMaximum(controlRef)) && scrollDistance < 0) ||
       ((controlValue == GetControlMinimum(controlRef)) && scrollDistance > 0))
      return;

    doMoveScrollBox(controlRef,scrollDistance);
  }

  NumToString((SInt32) GetControlValue(controlRef),valueString);
  doConcatPStrings(gCurrentString,valueString);
  windowRef = GetControlOwner(controlRef);
  doDrawMessage(windowRef,true);
}

// ************************************************************************
doMoveScrollBox

void doMoveScrollBox(ControlRef controlRef,SInt16 scrollDistance)
{
  SInt16 oldControlValue, controlValue, controlMax;

  oldControlValue = GetControlValue(controlRef);
```

```
  controlMax = GetControlMaximum(controlRef);

  controlValue = oldControlValue - scrollDistance;

  if(controlValue < 0)
    controlValue = 0;
  else if(controlValue > controlMax)
    controlValue = controlMax;

  SetControlValue(controlRef,controlValue);
}
// ********************************************************************************
doRadioButtons

void  doRadioButtons(ControlRef controlRef,WindowRef windowRef)
{
  docStrucHandle docStrucHdl;

  docStrucHdl = (docStrucHandle) (GetWRefCon(windowRef));

  SetControlValue((*docStrucHdl)->radiobuttonRedRef,kControlRadioButtonUncheckedValue);
  SetControlValue((*docStrucHdl)->radiobuttonWhiteRef,kControlRadioButtonUncheckedValue);
  SetControlValue((*docStrucHdl)->radiobuttonBlueRef,kControlRadioButtonUncheckedValue);
  SetControlValue(controlRef,kControlRadioButtonCheckedValue);
}

// ********************************************************************************
doCheckboxes

void  doCheckboxes(ControlRef controlRef)
{
  SetControlValue(controlRef,!GetControlValue(controlRef));
}

// ********************************************************************************
doPushButtons

void  doPushButtons(ControlRef controlRef,WindowRef windowRef)
{
  docStrucHandle docStrucHdl;

  docStrucHdl = (docStrucHandle) (GetWRefCon(windowRef));

  if(controlRef == (*docStrucHdl)->buttonRef)
  {
    doCopyPString("\pCancel button",gCurrentString);
    doDrawMessage(windowRef,true);
  }
  else if(controlRef == (*docStrucHdl)->buttonDefaultRef)
  {
    doCopyPString("\pDefault button",gCurrentString);
    doDrawMessage(windowRef,true);
  }
}

// ********************************************************************************
doAdjustScrollBars

void  doAdjustScrollBars(WindowRef windowRef)
{
  Rect           portRect;
  docStrucHandle docStrucHdl;

  docStrucHdl = (docStrucHandle) (GetWRefCon(windowRef));

  GetWindowPortBounds(windowRef,&portRect);
```

```
  HideControl((*docStrucHdl)->scrollbarVertRef);
  HideControl((*docStrucHdl)->scrollbarHorizRef);
  MoveControl((*docStrucHdl)->scrollbarVertRef,portRect.right - kScrollBarWidth,
              portRect.top + 25);
  MoveControl((*docStrucHdl)->scrollbarHorizRef,portRect.left -1,
              portRect.bottom - kScrollBarWidth);

  SizeControl((*docStrucHdl)->scrollbarVertRef,16, portRect.bottom - 39);
  SizeControl((*docStrucHdl)->scrollbarHorizRef, portRect.right - 13,16);

  ShowControl((*docStrucHdl)->scrollbarVertRef);
  ShowControl((*docStrucHdl)->scrollbarHorizRef);

  SetControlMaximum((*docStrucHdl)->scrollbarVertRef,portRect.bottom - portRect.top - 25
                    - kScrollBarWidth);
  SetControlMaximum((*docStrucHdl)->scrollbarHorizRef,portRect.right - portRect.left
                    - kScrollBarWidth);
}

// ****************************************************************************
doAdjustCursor

void  doAdjustCursor(WindowRef windowRef)
{
  RgnHandle  myArrowRegion;
  RgnHandle  myIBeamRegion;
  Rect       cursorRect;
  Point      mousePt;
  ControlRef controlRef;
  Cursor     arrow;

  myArrowRegion = NewRgn();
  myIBeamRegion = NewRgn();
  SetRectRgn(myArrowRegion,-32768,-32768,32767,32767);

  SetRect(&cursorRect,24,250,254,269);

  SetPortWindowPort(windowRef);
  LocalToGlobal(&topLeft(cursorRect));
  LocalToGlobal(&botRight(cursorRect));

  RectRgn(myIBeamRegion,&cursorRect);
  DiffRgn(myArrowRegion,myIBeamRegion,myArrowRegion);

  GetGlobalMouse(&mousePt);
  GetKeyboardFocus(FrontWindow(),&controlRef);

  if(PtInRgn(mousePt,myIBeamRegion) && controlRef)
    SetCursor(*(GetCursor(iBeamCursor)));
  else
    SetCursor(GetQDGlobalsArrow(&arrow));

  DisposeRgn(myArrowRegion);
  DisposeRgn(myIBeamRegion);
}
// ****************************************************************************
doDrawDocumentTyping

void  doDrawDocumentTyping(SInt8 charCode,UInt32 modifiers)
{
  Rect        typingRect  = {  118,128,194,253 } ;
  Rect        shiftRect   = {  131,181,139,189 } ;
  Rect        controlRect = {  144,181,152,189 } ;
  Rect        optionRect  = {  157,181,165,189 } ;
  Rect        cmdRect     = {  170,181,178,189 } ;
```

```
  Rect        textBoxRect;
  CFStringRef stringRef;

  EraseRect(&typingRect);

  stringRef = CFStringCreateWithPascalString(NULL,"\pShift",kCFStringEncodingMacRoman);
  SetRect(&textBoxRect,142,132,242,147);
  if((modifiers & shiftKey) != 0)  TextMode(srcOr); else TextMode(grayishTextOr);
  DrawThemeTextBox(stringRef,kThemeSmallSystemFont,0,false,&textBoxRect,teJustLeft,NULL);

  stringRef = CFStringCreateWithPascalString(NULL,"\pControl",kCFStringEncodingMacRoman);
  SetRect(&textBoxRect,142,145,242,160);
  if((modifiers & controlKey) != 0)  TextMode(srcOr); else TextMode(grayishTextOr);
  DrawThemeTextBox(stringRef,kThemeSmallSystemFont,0,false,&textBoxRect,teJustLeft,NULL);

  stringRef = CFStringCreateWithPascalString(NULL,"\pOption",kCFStringEncodingMacRoman);
  SetRect(&textBoxRect,142,158,242,173);
  if((modifiers & optionKey) != 0)  TextMode(srcOr); else TextMode(grayishTextOr);
  DrawThemeTextBox(stringRef,kThemeSmallSystemFont,0,false,&textBoxRect,teJustLeft,NULL);

  stringRef = CFStringCreateWithPascalString(NULL,"\pCmd",kCFStringEncodingMacRoman);
  SetRect(&textBoxRect,142,171,242,186);
  if((modifiers & cmdKey) != 0)  TextMode(srcOr); else TextMode(grayishTextOr);
  DrawThemeTextBox(stringRef,kThemeSmallSystemFont,0,false,&textBoxRect,teJustLeft,NULL);

  if(stringRef != NULL)
    CFRelease(stringRef);

  TextMode(srcOr);
  MoveTo(205,171);
  DrawChar(charCode);
}

// *****************************************************************************
doDrawMessage

void  doDrawMessage(WindowRef windowRef,Boolean inState)
{
  Rect        portRect, headerRect, textBoxRect;
  CFStringRef stringRef;

  SetPortWindowPort(windowRef);

  GetWindowPortBounds(windowRef,&portRect);
  SetRect(&headerRect,portRect.left - 1,portRect.top - 1,portRect.right + 1,
          portRect.top + 26);
  DrawThemeWindowHeader(&headerRect,inState);

  stringRef = CFStringCreateWithPascalString(NULL,gCurrentString,
                                             kCFStringEncodingMacRoman);
  SetRect(&textBoxRect,portRect.left,5,portRect.right,25);

  if(inState == kThemeStateActive)
    TextMode(srcOr);
  else
    TextMode(grayishTextOr);

  DrawThemeTextBox(stringRef,kThemeSmallSystemFont,inState,false,&textBoxRect,teJustCenter,
                   NULL);
  if(stringRef != NULL)
    CFRelease(stringRef);
}

// *****************************************************************************
doConcatPStrings
```

```
void  doConcatPStrings(Str255 targetString,Str255 appendString)
{
  SInt16 appendLength;

  appendLength = MIN(appendString[0],255 - targetString[0]);

  if(appendLength > 0)
  {
    BlockMoveData(appendString+1,targetString+targetString[0]+1,(SInt32) appendLength);
    targetString[0] += appendLength;
  }
}

// ************************************************************************
doCopyPString

void  doCopyPString(Str255 sourceString,Str255 destinationString)
{
  SInt16 stringLength;

  stringLength = sourceString[0];
  BlockMove(sourceString + 1,destinationString + 1,stringLength);
  destinationString[0] = stringLength;
}

//
**************************************************************************************************
// Dialogs.c
//
**************************************************************************************************

// ..........................................................................
includes

#include "CarbonEvents2.h"
// ..................................................................... global
variables

Boolean gSound   = 0;
Boolean gVideo   = 0;
Boolean gEffects = 0;

extern Str255 gCurrentString;

// ************************************************************************
doSheetAlert

OSStatus  doSheetAlert(void)
{
  AlertStdCFStringAlertParamRec paramRec;
  Str255                        messageText, informativeText;
  CFStringRef                   messageTextCF, informativeTextCF;
  OSStatus                      osError = noErr;
  DialogRef                     dialogRef;

  GetStandardAlertDefaultParams(&paramRec,kStdCFStringAlertVersionOne);
  paramRec.cancelText  = CFSTR("Cancel");
  paramRec.otherText  = CFSTR("Other");

  GetIndString(messageText,rSheetStrings,sAlertSheetMessage);
  GetIndString(informativeText,rSheetStrings,sAlertSheetInformative);
  messageTextCF = CFStringCreateWithPascalString(NULL,messageText,
                                                 CFStringGetSystemEncoding());
  informativeTextCF = CFStringCreateWithPascalString(NULL,informativeText,
                                                     CFStringGetSystemEncoding());
```

```
  osError = CreateStandardSheet(kAlertCautionAlert,messageTextCF,informativeTextCF,
&paramRec,
                                GetWindowEventTarget(FrontWindow()),&dialogRef);
  if(osError == noErr)
    osError = ShowSheetWindow(GetDialogWindow(dialogRef),FrontWindow());

  if(messageTextCF != NULL)
    CFRelease(messageTextCF);
  if(informativeTextCF != NULL)
    CFRelease(informativeTextCF);

  doAdjustMenus();

  return osError;
}

// ************************************************************************
doSheetDialog

OSStatus  doSheetDialog(void)
{
  DialogRef     dialogRef;
  WindowRef     windowRef;
  EventTypeSpec sheetEvents[] = {  kEventClassMouse, kEventMouseDown } ;
  ControlRef    controlRef;
  Str255        stringData = "\pBradman";
  OSStatus      osError = noErr;
  dialogRef = GetNewDialog(rSheetDialog,NULL,(WindowRef) -1);
  windowRef = GetDialogWindow(dialogRef);
  ChangeWindowAttributes(windowRef,kWindowStandardHandlerAttribute,0);

  InstallWindowEventHandler(windowRef,doGetSheetHandlerUPP(),GetEventTypeCount(sheetEvents),
                            sheetEvents,0,NULL);

  SetDialogDefaultItem(dialogRef,kStdOkItemIndex);

  GetDialogItemAsControl(dialogRef,2,&controlRef);
  SetDialogItemText((Handle) controlRef,stringData);
  SelectDialogItemText(dialogRef,2,0,32767);

  osError = ShowSheetWindow(GetDialogWindow(dialogRef),FrontWindow());

  doAdjustMenus();

  return osError;
}

// ************************************************************************
doGetSheetHandlerUPP

EventHandlerUPP  doGetSheetHandlerUPP(void)
{
  static EventHandlerUPP sheetEventHandlerUPP;

  if(sheetEventHandlerUPP == NULL)
    sheetEventHandlerUPP = NewEventHandlerUPP((EventHandlerProcPtr) sheetEventHandler);

  return sheetEventHandlerUPP;
}

// ************************************************************************
sheetEventHandler

OSStatus  sheetEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                            void* userData)
{
```

```
  OSStatus        result = eventNotHandledErr;
  UInt32          eventClass;
  UInt32          eventKind;
  Point           mouseLocation;
  ControlRef      controlRef, controlRefOKButton;
  ControlPartCode controlPartCode;
  DialogRef       dialogRef;
  Rect            portRect;

  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);

  if(eventClass == kEventClassMouse)
  {
    if(eventKind == kEventMouseDown)
    {
      GetEventParameter(eventRef,kEventParamMouseLocation,typeQDPoint,NULL,
                        sizeof(mouseLocation),NULL,&mouseLocation);
      SetPortWindowPort(FrontWindow());
      GlobalToLocal(&mouseLocation);
      controlRef = FindControlUnderMouse(mouseLocation,FrontWindow(),&controlPartCode);
      if(controlRef)
      {
        dialogRef = GetDialogFromWindow(FrontWindow());
        GetDialogItemAsControl(dialogRef,1,&controlRefOKButton);

        if(controlRef == controlRefOKButton)
        {
          GetDialogItemAsControl(dialogRef,2,&controlRef);
          GetDialogItemText((Handle) controlRef,gCurrentString);

          HideSheetWindow(FrontWindow());
          DisposeDialog(dialogRef);

          GetWindowPortBounds(FrontWindow(),&portRect);
          InvalWindowRect(FrontWindow(),&portRect);

          return noErr;
        }
      }
    }
  }

  return result;
}

// ********************************************************************
doMovableModalDialog

OSStatus  doMovableModalDialog(void)
{
  OSStatus      osError = noErr;
  Rect          rect = {  0,0,167,148 } ;
  WindowRef     windowRef;
  Rect          pushButtonRect = {  127,63,147,132 } ;
  ControlRef    controlRef;
  ControlRef    soundControlRef, videoControlRef, effectsControlRef, boxControlRef;
  Rect          checkboxRect = {  37,32,55,124 } ;
  ControlID     controlID;
  Rect          groupboxRect = {  10,16,113,132 } ;
  EventTypeSpec dialogEvents[] = {  kEventClassControl, kEventControlHit } ;

  osError = CreateNewWindow(kMovableModalWindowClass,kWindowStandardHandlerAttribute,&rect,
                            &windowRef);
  if(osError == noErr)
  {
```

```
    RepositionWindow(windowRef,FrontWindow(),kWindowAlertPositionOnMainScreen);
    SetThemeWindowBackground(windowRef,kThemeBrushDialogBackgroundActive,false);

    CreateRootControl(windowRef,&controlRef);

    CreatePushButtonControl(windowRef,&pushButtonRect,CFSTR("OK"),&controlRef);
    SetWindowDefaultButton(windowRef,controlRef);
    controlID.id = 'okbt';
    SetControlID(controlRef,&controlID);
    CreateCheckBoxControl(windowRef,&checkboxRect,CFSTR("Sound On"),1,false,
&soundControlRef);
    controlID.id = 'chb1';
    SetControlID(soundControlRef,&controlID);
    SetControlValue(soundControlRef,gSound);

    OffsetRect(&checkboxRect,0,22);
    CreateCheckBoxControl(windowRef,&checkboxRect,CFSTR("Video On"),1,false,
&videoControlRef);
    controlID.id = 'chb2';
    SetControlID(videoControlRef,&controlID);
    SetControlValue(videoControlRef,gVideo);
    OffsetRect(&checkboxRect,0,22);
    CreateCheckBoxControl(windowRef,&checkboxRect,CFSTR("Effects On"),1,false,
                          &effectsControlRef);
    controlID.id = 'chb3';
    SetControlID(effectsControlRef,&controlID);
    SetControlValue(effectsControlRef,gEffects);

    CreateGroupBoxControl(windowRef,&groupboxRect,CFSTR("Preferences"),true,&boxControlRef);

    AutoEmbedControl(soundControlRef,windowRef);
    AutoEmbedControl(videoControlRef,windowRef);
    AutoEmbedControl(effectsControlRef,windowRef);

    InstallWindowEventHandler(windowRef,doGetDialogHandlerUPP(),
                              GetEventTypeCount(dialogEvents),dialogEvents,windowRef,NULL);
    ShowWindow(windowRef);
    osError = RunAppModalLoopForWindow(windowRef);
  }

  return osError;
}

// ************************************************************************
doGetDialogHandlerUPP

EventHandlerUPP  doGetDialogHandlerUPP(void)
{
  static EventHandlerUPP dialogEventHandlerUPP;

  if(dialogEventHandlerUPP == NULL)
    dialogEventHandlerUPP = NewEventHandlerUPP((EventHandlerProcPtr) dialogEventHandler);

  return dialogEventHandlerUPP;
}

// ************************************************************************
dialogEventHandler

OSStatus  dialogEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                             void *userData)
{
  OSStatus   result = eventNotHandledErr;
  UInt32     eventClass;
  UInt32     eventKind;
  ControlRef controlRef;
```

```
  ControlID  controlID;
  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);

  if(eventClass == kEventClassControl)
  {
    if(eventKind == kEventControlHit)
    {
      GetEventParameter(eventRef,kEventParamDirectObject,typeControlRef,NULL,
                        sizeof(ControlRef),NULL,&controlRef);

      GetControlID(controlRef,&controlID);
      if(controlID.id == 'okbt')
      {
        QuitAppModalLoopForWindow(userData);
        DisposeWindow(userData);
        result = noErr;
      }
      else
      {
        SetControlValue(controlRef,!GetControlValue(controlRef));
        if(controlID.id == 'chb1')
          gSound = GetControlValue(controlRef);
        else if(controlID.id == 'chb2')
          gVideo = GetControlValue(controlRef);
        else if(controlID.id == 'chb3')
          gEffects = GetControlValue(controlRef);
        result = noErr;
      }
    }
  }
  return result;
}

//
*********************************************************************************************
```

Demonstration Program CarbonEvents2 Comments

When this program is run, the user should

- Open and close windows, and drag, resize, and zoom open windows, noting particularly the size to which the window zooms in an out.
- Interact with the pop-up menu button, push button, and scrollbar controls in open windows.
- Send the application to the background and bring it to the foreground, noting the activation and deactivation of the window controls.
- Type into the edit text control, with and without the Shift, Control, Option, and/or Command keys held down. Then choose Document or Edit Text Control to change the target for keyboard input.
- Choose the items in the Dialogs menu to open, close, and interact with the movable modal dialog and, on Mac OS X only, window-modal (sheet) alerts and window-modal (sheet) dialogs.
- Quit the application by choosing the Quit item in the Mac OS 8/9 File/Mac OS X Application menu and using its Command-key equivalent.

The functions relating to controls in this program, including the action functions for the scroll bars, are similar to those in the demonstration programs Controls1 and Controls2 (Chapter 7).

CarbonEvents2.c

main

If the program is running on OS 8/9, `SetMenuItemCommandID` is called to assign the command ID 'quit' to the Quit item in the File menu. (This command is assigned to the Mac OS X Quit item by default.) Thus, when the Quit item is chosen on Mac OS 8/9 and Mac OS X, the standard application event handler will call the default Quit Application Apple event handler (automatically installed when `RunApplicationEventLoop` is called) to close down the program.

The call to `InstallApplicationEventHandler` installs the program's application

event handler.

The call to `InstallEventLoopTimer` installs a timer set to fire at the interval returned by the call to `GetCaretTime`, which is converted to event time (seconds) by the macro `TicksToEventTime`. The timer will be used to trigger a call to the function `doIdle`, within which `IdleControls` is called to blink the insertion point caret in the windows' edit text control. A universal procedure pointer to doIdle is passed in the `inTimerProc` parameter.

When `RunApplicationEventLoop` is called, registered events will be dispatched to the application.

appEventHandler

`appEventHandler` is the application's application event handler. It is a callback function.

The calls to `GetEventClass` and `GetEventKind` get the event class and type. The function then switches on the event class.

If the event class is `kEventClassApplication` and the event type is `kEventAppActivated`, the cursor is set to the arrow cursor. `eventNotHandledErr` is returned by the handler, ensuring that the event will be propagated to the standard application event handler.

If the event class is `kEventClassCommand` and the event type is `kEventProcessCommand`, `GetEventParameter` is called to extract the specified data from the event. This data is returned in a variable of type `HICommand`.

If an examination of the `commandID` field of the `HICommand` structure reveals that the command ID is 'quit', the handler returns eventNotHandledErr, ensuring that the event will be propagated to the standard application event handler. This ensures that the standard handler calls the default Quit Application Apple event handler.

The menu ID and item number are then extracted from the `HICommand` structure. If the command ID is not 'quit' and the menu ID is that for one of the program's pull-down menus, `doMenuChoice` is called to further handle the event. Because the event is fully handled by the program, `noErr` is returned by the handler to ensure that the event is not further propagated.

In this program, each pop-up menu button control is handled in a different manner. As will be seen, the second (Country) pop-up menu button control is assigned the command ID `'ctry'` (`kPopupCountryID`) on creation. Thus, if a mouse-down occurs in this pop-up menu button, the standard window event handler calls `TrackControl` to handle user

action, following which the kEventProcessCommand is dispatched to the application. Within the application's application event handler, if the command ID in the HICommand structure's commandID field is 'ctry', the function doControlHit2 is called, following which noErr is returned by the handler.

If the event class is kEventClassMenu and the event type is kEventMenuEnableItems, the function doAdjustMenus is called. The kEventMenuEnableItems event type is dispatched when a mouse-down occurs in a pull-down menu or a menu-related Command-key equivalent is pressed.

If the event class is kEventClassMouse and the event type is kEventMouseMoved, and if the front window is of the document kind, the function doAdjustCursor is called to adjust the cursor to the I-beam shape if it is over an edit text control with keyboard focus, or to the arrow shape if it is not.

windowEventHandler

windowEventHandler is the application's window event handler. It is a callback function.

The calls to GetEventClass and GetEventKind get the event class and type. The function then switches on the event class.

If the event class is kEventClassWindow, GetEventParameter is called to extract the window reference from the event before a switch on the event type is entered.

If the event type is kEventWindowDrawContent, the function doDrawContent (the window update function) is called. (Note that doDrawContent does not call BeginUpdate and EndUpdate because there is no need to call those functions when responding to kEventWindowDrawContent events.) The handler returns eventNotHandledErr, allowing the event to be passed to the standard window event handler which, in turn, attends to its part of the update process, including drawing the controls.

Note that registering the kEventWindowDrawContent event type and responding in this way obviates the necessity for an event filter (callback) function (which calls the window update function) for the movable modal dialog.

If the event type is kEventWindowActivated or kEventWindowDeactivated, the function doActivateDeactivate is called to draw the text in the window header in the appropriate (activated or deactivated) state. The handler returns eventNotHandledErr, allowing the event to be passed to the standard window event handler which, in turn, attends to its part of the activation/deactivation process, including activating/deactivating the controls.

If the event type is `kEventWindowGetIdealSize`, the handler responds by calling `SetEventParameter`, which sets the height and width to which the window will be zoomed when it is zoomed out. The handler returns `noErr` to defeat further propagation of the event.

If the event type is `kEventWindowGetMinimumSize`, the handler responds by calling `SetEventParameter`, which sets the minimum height and width to which the window can be resized. The handler returns `noErr` to defeat further propagation of the event.

If the event type is `kEventWindowZoomed`, the window's port rectangle is erased, the function `doAdjustScrollBars` is called to resize and reposition the scroll bars, and the handler returns `noErr`.

Since the `kWindowLiveResizeAttribute` attribute is set on the window, the `kEventWindowBoundsChanged` event type will be received continually as the window is being resized (as opposed to only one `kEventWindowBoundsChanged` event type being received, when the mouse button is released, when the `kWindowLiveResizeAttribute` attribute is not set). The function `doAdjustScrollBars` is continally called to resize and reposition the scroll bars, the function `doDrawMessage` is continually called to redraw the window header and associated text, and the handler returns `noErr`.

If the event type is `kEventWindowClose`, the function `doCloseWindow` is called to dispose of the window's controls and document structure handle, decrement the global variable holding the current number of open windows, and disable the Typing Target and Window menus if no windows will be open when this window is closed. `EventNotHandledErr` is returned by the handler to cause the standard window event handler to dispose of the window.

If the event class is `kEventClassControl` and the event type is `kEventControlClick`, `GetEventParameter` is called to extract the mouse location, which will be in global coordinates, from the event. The mouse coordinates are then converted to local coordinates preparatory to a call to `FindControlUnderMouse`. If there is a control under the mouse cursor, the function `doControlHit1` is called to further handle the event. When `doControlHit1` returns, the handler returns `noErr` to defeat further propagation of the event. (Note that, so far as the first (Time Zone) pop-up menu button is concerned, this approach to pop-up menu button control handling differs from that in the demonstration program CarbonEvents1.)

If the event class is `kEventClassKeyboard` and the event type is `kEventRawKeyDown`, and if the current typing target (as chosen in the Typing Target menu) is either the "document" or the "document" and edit text control combined, `GetEventParameter` is called twice to get the character code and the modifier keys that were down (if any), and both of these parameters are passed to the function

doDrawDocumentTyping to draw the character and highlight the modifier key (if any) indicator in the ("document") Typing group box in the window. If the current typing target is the "document" only, the handler returns noErr to prevent the edit text control receiving the event, otherwise eventNotHandledErr is returned to allow the event to propagate to the edit text control.

The last block pertains to the window-modal (sheet) alert created by the function doSheetAlert in Dialogs.c. When the user clicks in one of the sheet's buttons, the parent window receives the relevant command ID (kHICommandOK, kHICommandCancel, or kHICommandOther). The identity of the button is drawn in the parent window's window header frame.

doNewWindow

After GetNewCWindow creates the window, ChangeWindowAttributes is called to cause the standard window event handler to be installed on the window.

The next block shows alternative window creation code for windows created programmatically using CreateNewWindow. Note that the standard window event handler will be installed because the kWindowStandardHandlerAttribute is included in the attributes passed in the second parameter of the CreateNewWindow call.

The call to InstallWindowEventHandler installs the application's window event handler on the window. Since more than one window can be opened, the call to InstallWindowEventHandler will be called whenever a new window is opened. Accordingly, to prevent a possible memory leak, the call to doGetHandlerUPP (to get a UPP to the application's window event handler) ensures that only one routine descriptor will be created regardless of how many windows are opened. (Recall from Chapter 5 that universal procedure pointer creation functions always allocate routine descriptors in memory on Mac OS 8/9, and sometimes allocate routine descriptors in memory on Mac OS X (depending on whether the application is compiled as a CFM binary or Mach-O binary.)

Note that registering the kEventProcessCommand event (class kEventClassCommand) is required in order to determine which button is clicked in a window-modal (sheet) alert.

doCloseWindow

Note that doCloseWindow does not dispose of the window. As previously stated, the application's window event handler allows kEventWindowClose events to be passed to the standard window event handler after doCloseWindow is called. The standard window handler disposes of the window.

doGetControls

After the controls have been created, `SetControlCommandID` is called to assign the command ID `'ctry'` (`kPopupCountryID`) to the second (Country) pop-up menu button control. As previously stated, this will cause a `kEventProcessCommand` event to be dispatched to the application when a a mouse-down occurs in this pop-up menu button, allowing the application's application event handler to handle the event.

doIdle

`doIdle` is called when the event timer fires. `IdleControls` is called to cause the insertion point caret to blink in the edit text control. (This call is not necessary on Mac OS X because Mac OS X controls have built-in timers.)

doDrawContent

`doDrawContent` is the window update function, which is called when the `kEventWindowDrawContent` event type is received. As previously stated, there is no need to call `BeginUpdate` or `EndUpdate` in this function (though there would be, for Mac OS 8/9 only, if the `kEventWindowUpdate` event type had been registered rather than the `kEventWindowDrawContent` event type).

doControlHit1

`doControlHit1` is called from the application's window event handler when the `kEventMouseDown` event type is received and a call to `FindControlUnderMouse` determines that there is a control under the mouse. Further processing of mouse-downs in the controls in this program is identical to that used in in the demonstration program Controls1 (see Chapter 7).

Note the differences in this program's approach to detecting and handling mouse-downs in a pop-up menu button control, as compared with the approach used in the demonstration program CarbonEvents1.

doControlHit2

`DoControlHit2` is called from the application's application event handler when the `kEventProcessCommand` event type is received and the commandID field of the `HICommand` structure contains the second (Country) pop-up menu button control's command ID. The control's value is determined, allowing the chosen menu item's text to be extracted and drawn in the window header.

Dialogs.c

doSheetAlert

The call to `GetStandardAlertDefaultParams` initializes a standard `CFString` alert parameter structure with default values. (The defaults are: no Help button; no Cancel button; no Other button.) The next two lines cause a Cancel and Other button to be included.

The next block gets the message and informative strings to be passed in the call to `CreateStandardSheet`, which creates the sheet. The call to `ShowSheetWindow` displays the sheet.

Clicks in the sheet's buttons are handled in the parent window's handler (`windowEventHandler`).

doSheetDialog

The call to `GetNewDialog` creates the dialog, whose window definition ID is `kWindowSheetProc` (1088). `GetDialogWindow` gets a reference to the dialog's window object, allowing `ChangeWindowAttributes` to be called to cause the standard window event handler to be installed on the window.

The call to `InstallWindowEventHandler` installs the handler sheetEventHandler on the dialog. Note that the function `doGetSheetHandlerUPP` is called to get the universal procedure pointer to the handler passed in the second parameter of the `InstallWindowEventHandler` call.

`SetDialogDefaultItem` establishes the single push button item in the dialog as the default button. The next block sets some initial text in the dialog's edit text item and selects that text.

The call to `ShowSheetWindow` displays the sheet.

doGetSheetHandlerUPP

`doGetSheetHandler` serves the same purpose for window-modal (sheet) dialogs as does `doGetHandlerUPP` (see above) for document windows.

sheetEventHandler

`sheetEventHandler` is the event handler for window-modal (sheet) dialogs. It is a callback function.

If the `kEventMouseDown` event type is received, `GetEventParameter` is called to get the mouse location, which is then converted to the local coordinates required by `FindControlUnderMouse`. If the call to `FindControlUnderMouse` reveals that

there is a control under the mouse, `GetDialogFromWindow` is called to get a reference to the dialog, allowing `GetDialogItemAsControl` to get a reference to the first item in the dialog's item list (the OK push button).

If the control clicked is the OK push button, the current text in the edit text item is extracted for display in the parent window's window header frame, `HideSheetWindow` is called to hide the sheet and `DisposeDialog` disposes of the sheet and releases all related memory.

If the mouse click was in the edit text item, `eventNotHandledErr` is returned so that the event can be handled by the standard handler.

doMovableModalDialog

`doMovableModalDialog` creates a movable modal dialog containing three checkbox controls, a group box control, and an OK push button control.

`CreateNewWindow` creates an initially invisible window of class `kMovableModalWindowClass` with the standard window handler installed. The RepositionWindow call ensures that the dialog will appear in the alert position on the main screen. `SetThemeWindowBackground` sets the dialog's background color/pattern to the correct color pattern for dialogs.

`CreateRootControl` creates a root control for the window, ensuring that activation/deactivation of the controls will be automatic.

The next three blocks create the dialog's controls. In the case of the OK push button and checkbox controls, a control ID is assigned to each control. The initial value of the checkbox controls is set to 0.

`InstallWindowEventHandler` installs the event handler dialogEventHandler on the window. With the dialog fully prepared, `ShowWindow` displays the dialog.

`RunAppModalLoopForWindow` is the Carbon event model equivalent of the Classic event model's `ModalDialog`. It will exit when `QuitAppModalLoopForWindow` is called in the dialog's event handler. Although it will block until the modal loop ends, your application's other handlers will still be called.

`RunAppModalLoopForWindow` attends to the menu deactivation usually associated with the display of a movable modal dialog.

dialogEventHandler

`dialogEventHandler` is the event handler for the movable modal dialog. It is a

callback function.

If the `kEventControlHit` event type is received, `GetEventParameter` is called to get a reference to the control, allowing the call to `GetControlID` to get the control's ID.

If the control is the OK push button, the `QuitAppModalLoopForWindow` call terminates the modal loop and restores menu activation/deactivation status to that which obtained prior to the call to `RunAppModalLoopForWindow`.

If the control is one of the checkbox controls, the current value of the control is flipped and the new value is assigned to the relevant global variable which keeps track of the control values between successive invocations of the dialog.

Chapter 18. Files and Navigation Services

IN THIS CHAPTER

- Introduction
- Files
- Navigation Services
- Main File Manager Constants, Data Types and Functions
- Relevant Resource Manager Functions
- Relevant Finder Interface Functions
- Main Navigation Services Constants, Data Types, and Functions
- Demonstration Program Files Listing
- Demonstration Program Files Comments

Introduction

This chapter addresses

- Creating, opening, reading from, writing to, and closing files.
- Navigation Services, an application programming interface that allows your application to provide a user interface for navigating, opening, and saving Mac OS file objects.

Files

Types of Files

A file is a named, ordered sequence of bytes stored on a volume. The files associated with an application are typically:

- The application file itself, which comprises the application's executable code and any application-specific resources and data.
- Document files created by the user, which the user can edit.
- A preferences file created by the application to store user-specified preference settings for the application.

The operating system also uses files for certain purposes. For example, as stated in Chapter 9, the File Manager uses a special file called the volume's **catalog file** to maintain the hierarchical organization of files and folders in a volume.

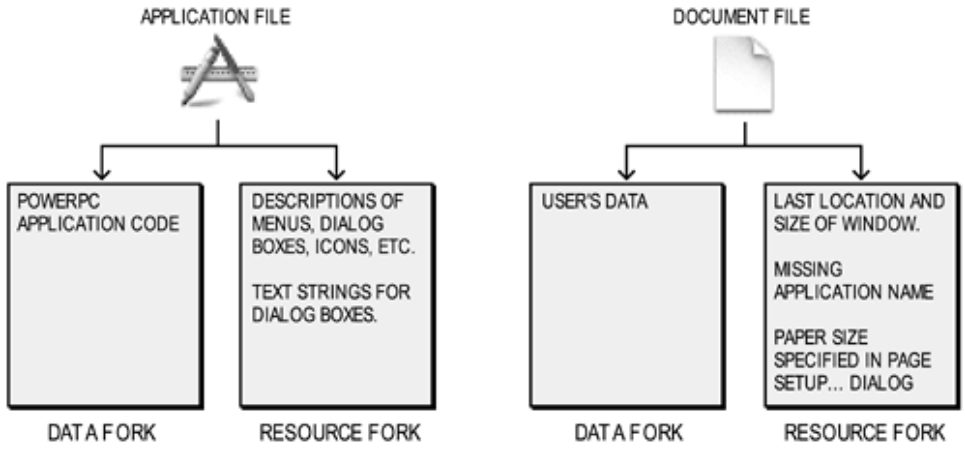
Characteristics of Files

File Forks

Macintosh files comprise two **forks,** called the **data fork** and the **resource fork.** The resource fork contains a resource map and resources. Unlike the bytes stored in the resource fork, the bytes in the data fork do not require any particular internal structure. Your application must therefore be able to interpret the bytes in the data fork in an appropriate manner.

All Macintosh files contain a data fork and a resource fork; however, one or both of these forks may be empty. Fig 1 shows the typical contents of the data and resource forks of an application file and a document file.

Figure 1. TYPICAL CONTENTS OF DATA FORKS AND RESOURCE FORKES IN APPLICATION AND DOCUMENT FILES



If your data can be structured as a resource, you might elect to store that data in the resource fork, in which case you use Resource Manager functions to both store and retrieve it. Retrieving data from a resource fork is a comparatively simple matter because all you have to do is pass the resource type and ID to the relevant Resource Manager function.

If it is neither possible nor advisable to store the data in the resource fork; you must store it in the data fork. This is normally the favored option for storing a document's text, for example. In this case, you use File Manager functions to store and retrieve the data. With File Manager functions, unlike Resource Manager functions, you can access any byte, or group of bytes, individually.

Generally speaking, unless the data created by the user will occupy only a small number of resources, you should store it in the data fork. The Resource Manager was not designed as a general purpose data storage and retrieval system.

File Size

Volumes

A **volume,** which can be an entire disk or only part thereof, is that part of a storage device formatted to contain files. Ordinarily, file size is limited only by the size of the volume that contains it.

Logical Blocks and Allocation Blocks

Volumes are formatted into **logical blocks.** Each logical block can contain up to 512 bytes, the actual size being of interest only to the disk device driver. When the File Manager allocates space for a file, it allocates it in units called **allocation blocks,** which are groups of consecutive logical blocks. A non-empty file fork always occupies at least one allocation block.

The size of an allocation block is the chief distinguishing feature between the volume format known as the Hierarchical File System (HFS) and the newer Hierarchical File System Plus (HFS Plus or HFS+) introduced with Mac OS 8.1. The differences are as follows:

- ***HFS (Mac OS Standard Format).*** For HFS-formatted volumes, the File Manager can access a maximum of 65,535 allocation blocks on any volume. Thus the larger the volume, the larger the allocation block. For example, on a 500 MB volume, the allocation block size is 8KB under HFS.

- ***HFS Plus (Mac OS Extended Format).*** For HFS Plus-formatted volumes, the File Manager can access a maximum of 4.29 billion allocation blocks on any volume. This means that even huge volumes can be formatted with very small allocation blocks. The default volume format for Carbon is HFS Plus.

  Note

  Beginning with Mac OS 9, HFS Plus introduced support for long Unicode filenames, files larger than 2GB, and extended file attributes. The additional File Manager constants, data types, and functions introduced at that time are often referred to as the HFS Plus API.

On large volumes, the significant reduction in allocation block size under HFS Plus results in significant space savings. For example, on a 4 GB volume, a file containing only 4 KB of information requires 64 KB of space under HFS, whereas the same file requires only 4KB of space under HFS Plus.

Physical and Logical End-Of-File

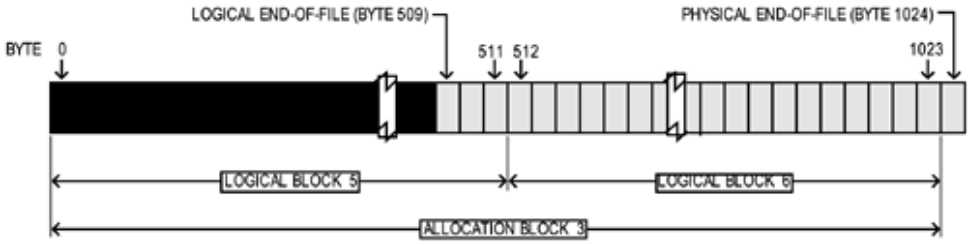
There is a difference between the amount of space allocated to a file and the number of bytes of actual data in the file. This is reflected in the two numbers used to describe the size of a file:

- ***Physical End-Of-File.*** The physical end-of-file is the number of bytes currently allocated to the file by the File Manager. Since the file's first byte is byte number 0, the physical end-of-file is 1 greater than the number of the last byte in its last allocation block. The physical end-of-file is thus always an exact multiple of allocation block size.

- ***Logical End-Of-File.*** The logical end-of-file is one greater than the number of bytes that currently contain data.

Fig 2 illustrates logical end-of-file and physical end-of-file.

Figure 2. LOGICAL AND PHYSICAL END-OF-FILE



Your application can adjust the size of a file by moving the logical end-of-file. If, when you increase the size of a file, the logical end-of-file is moved past the physical end-of-file, one or more allocation blocks are automatically added to the file by the File Manager. By the same token, the File Manager automatically deletes the unneeded allocation block if you move the logical end-of-file more than one allocation block short of the current physical end-of-file.

Clumps and Combating File Fragmentation

The volume's **clump** size determines the number of allocation blocks added to the file when you move the logical end-of-file past the physical end-of-file. The File Manager enlarges files by adding clumps (which are groups of *contiguous* allocation blocks) as a way of reducing file fragmentation and improving input/output performance.

Your application can also takes steps to reduce file fragmentation. Suppose you are extending a file with multiple write operations. If you know before you begin how large the file is likely to become, you should first call `SetEOF` to set the file to that size.

File Access

The operations your application can perform on a file depend on whether it is open or closed. For example, reading and writing operations can only be performed on open files, and deleting operations can only be performed on closed files.

Access Path and File Reference Number

When a file is opened, the File Manager reads in file information and creates an **access path** to the file. The file information is stored in a **file control block** (FCB). The access path, which is assigned a unique **file reference number,** specifies the volume on which the file is located and the location of the file on that volume.

File Mark

The File Manager maintains a **file mark** (a current-position marker) for each access path. The file mark, which is moved each time a byte is read or written, is the number of the next byte to be read or written. By setting the file mark or specifying an offset, you can control the beginning point of a read or write operation.

Data Buffer

When it transfers data to or from your application, the File Manager uses a **data buffer** in RAM. You must therefore pass the address of this data buffer whenever you read or write a file's data.

Disk Cache

The File Manager uses an intermediate buffer, called the **disk cache,** when reading data from, or writing data to, the file system.

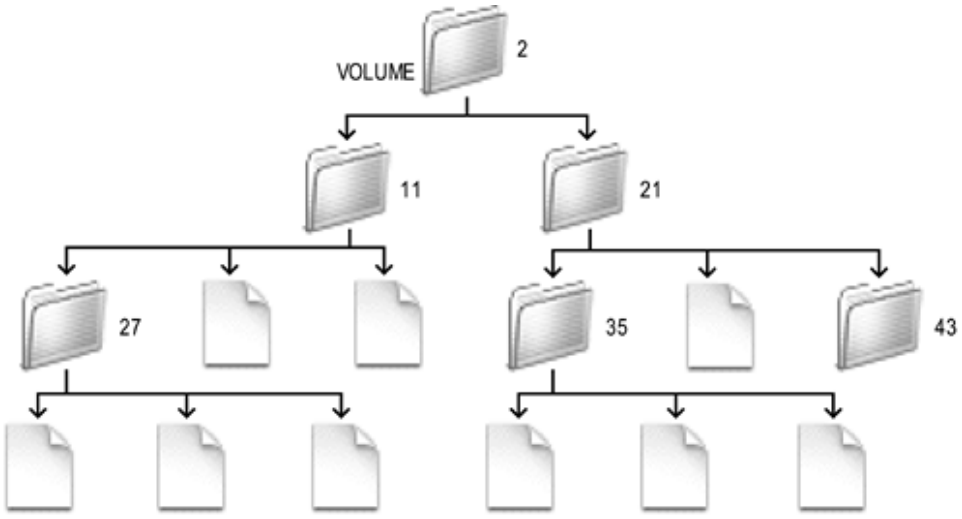
During a write operation, data is transferred from your application's data buffer to the disk cache. During a read operation, the File Manager looks for data in the disk cache and, if data is found in the cache, transfers that data to your application's data buffer. If the File Manager finds no data in the disk cache, it reads the requested number of bytes from the disk directly to your application's data buffer.

The Hierarchical File System

Directories and Directory ID

The method used to organize files on a Macintosh volume is called a **hierarchical file system.** In this system, files are grouped into **directories** (also called **folders**). These directories may, in turn, be grouped into other directories (see Fig 3). As shown in Fig 3, each directory has a number associated with it called the **directory ID.**

Figure 3. MACINTOSH HIERACHICAL FILE SYSTEM



Root Directory

The Finder and the File Manager work together to maintain the organization of files and folders on a volume, ensuring that the representation on the desktop corresponds directly to the hierarchical directory structure on the volume. In file system parlance, the volume is referred to as the **root directory,** and the folders are referred to as **subdirectories** (or simply as **directories**).

Mounted Volumes

When a volume is **mounted,** the File Manager places information about the volume in a **volume control block** (**VCB**) and assigns a **volume reference number** by which you can refer to the volume until it is **unmounted.** Mounted volumes appear on the desktop.

You can identify a volume by its volume reference number or by its **volume name.** To avoid confusion between volumes with the same name, you should ordinarily use the volume reference number to refer to a volume.

When a volume is unmounted, the volume control block is released and the volume is no longer known to the File Manager.

Parent Directory and Parent Directory ID

The directory in which a subdirectory is located is referred to as a **parent directory.** A parent directory is assigned a **parent directory ID.** A special parent directory ID is assigned by the File Manager to a volume's root directory. All this facilitates a consistent method of identifying files and directories using the volume reference number, the parent directory ID, and the file or directory name.

Generally speaking, your application does not need to keep track of the location of files in the file system hierarchy. The location of most of the files your application opens and saves is provided by either the Finder or Navigation Services.

Aliases

An **alias** is a special kind of file that represents another file, folder, or volume. The Finder and Navigation Services automatically resolve aliases.

Identifying Files and Directories-File System Specification Structure and File System Reference

Three pieces of information are all that is needed to identify a file or directory: a volume reference number; a parent directory ID; the name of the file or directory. Of relevance in this regard are two data types, namely, the **file system specification structure** and the opaque **file system reference:**

```
struct FSSpec
{
  short vRefNum;  // Volume reference number.
  long  parID;    // Directory ID of parent directory.
  Str63 name;     // Filename or directory name.
} ;
typedef struct FSSpec FSSpec;
typedef FSSpec *FSSpecPtr, **FSSpecHandle;

struct FSRef
{
  UInt8 hidden[80];
} ;
typedef struct FSRef FSRef;
typedef FSRef *FSRefPtr;
```

The opaque data type `FSRef`, whose purpose is similar to that of the file system specification structure, was introduced with the HFS Plus API. Note that there is no need to call the File Manager to dispose of an `FSRef` when it is no longer needed.

Creating, Opening, Reading From, Writing To, and Closing Files

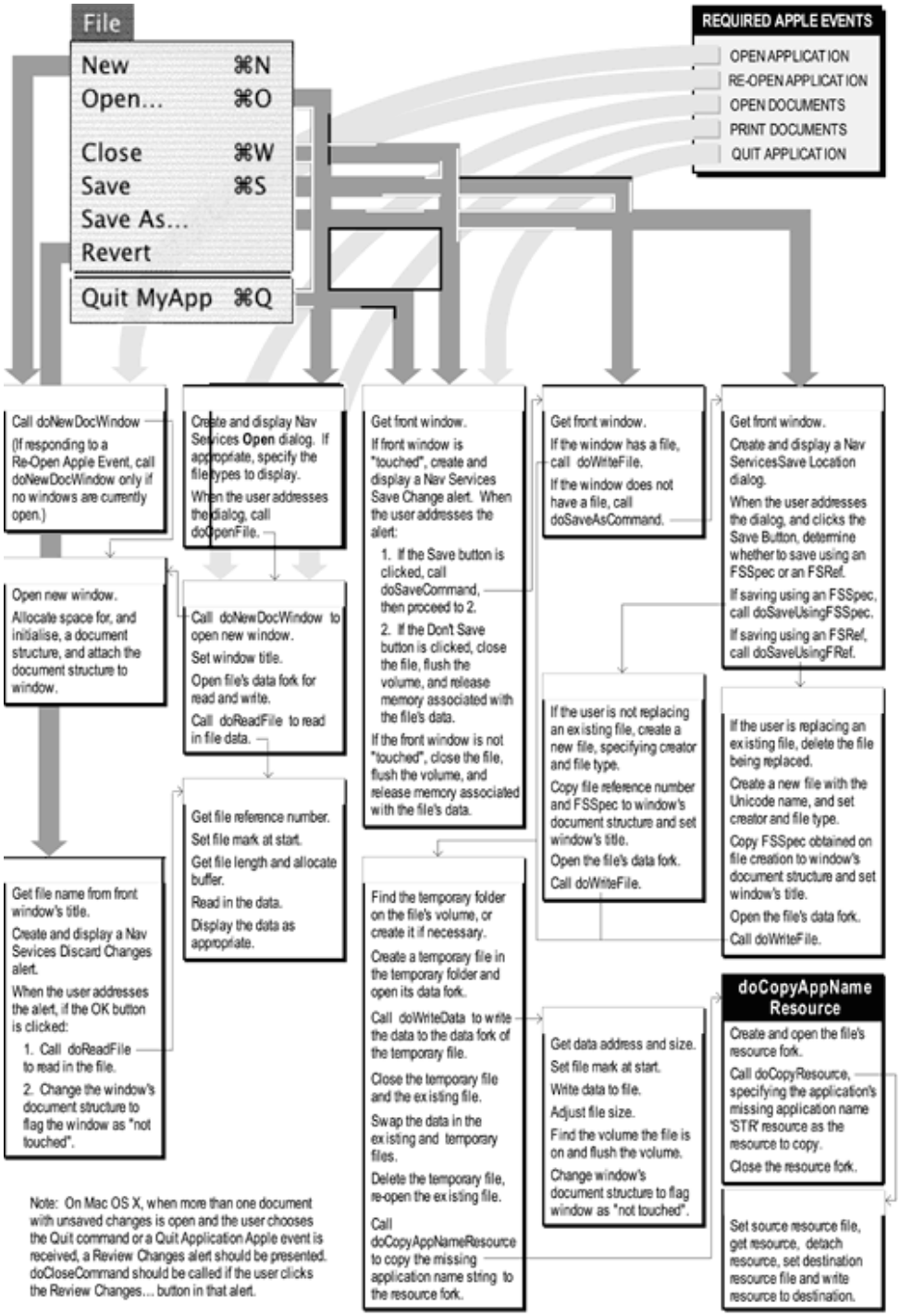
Your application typically creates, opens, reads from, writes to, and closes files in response to the user choosing commands from the File menu. In addition, your application opens, reads from, writes to, and closes files in response to the required Apple events (see Chapter 10).

The following describes how to perform typical file operations within the context of a user choosing commands from an imaginary application's File menu and, on Mac OS X, the Quit command. For the purposes of illustration, the assumption is made that the files involved store text documents and that, when retrieved from file, the documents are displayed in a window with scroll bars.

General File Menu and Required Apple Events Handling Strategy

A suggested general strategy for handling user choices of the New, Open, Close, Save, Save As, Revert, and Quit commands, and for responding to the required Apple events, is illustrated in Fig 4.

Figure 4. GENERAL FILE MENU, QUIT ITEM, AND REQUIRED APPLE EVENTS HANDLING STRATEGY


File
New ⌘N
Open... ⌘O
Close ⌘W
Save ⌘S
Save As...
Revert
Quit MyApp ⌘Q
REQUIRED APPLE EVENTS
OPEN APPLICATION
RE-OPEN APPLICATION
OPEN DOCUMENTS
PRINT DOCUMENTS
QUIT APPLICATION
Call doNewDocWindow (If responding to a Re-Open Apple Event, call doNewDocWindow only if no windows are currently open.)
Create and display Nav Services Open dialog. If appropriate, specify the file types to display. When the user addresses the dialog, call doOpenFile.
Get front window. If front window is "touched", create and display a Nav Services Save Change alert. When the user addresses the alert: 1. If the Save button is clicked, call doSaveCommand, then proceed to 2. 2. If the Don't Save button is clicked, close the file, flush the volume, and release memory associated with the file's data. If the front window is not "touched", close the file, flush the volume, and release memory associated with the file's data.
Get front window. If the window has a file, call doWriteFile. If the window does not have a file, call doSaveAsCommand.
Get front window. Create and display a Nav ServicesSave Location dialog. When the user addresses the dialog, and clicks the Save Button, determine whether to save using an FSSpec or an FSRef. If saving using an FSSpec, call doSaveUsingFSSpec. If saving using an FSRef, call doSaveUsingFRef.
Open new window. Allocate space for, and initialise, a document structure, and attach the document structure to window.
Call doNewDocWindow to open new window. Set window title. Open file's data fork for read and write. Call doReadFile to read in file data.
Get file reference number. Set file mark at start. Get file length and allocate buffer. Read in the data. Display the data as appropriate.
If the user is not replacing an existing file, create a new file, specifying creator and file type. Copy file reference number and FSSpec to window's document structure and set window's title. Open the file's data fork. Call doWriteFile.
If the user is replacing an existing file, delete the file being replaced. Create a new file with the Unicode name, and set creator and file type. Copy FSSpec obtained on file creation to window's document structure and set window's title. Open the file's data fork. Call doWriteFile.
Get file name from front window's title. Create and display a Nav Sevices Discard Changes alert. When the user addresses the alert, if the OK button is clicked: 1. Call doReadFile to read in the file. 2. Change the window's document structure to flag the window as "not touched".
Find the temporary folder on the file's volume, or create it if necessary. Create a temporary file in the temporary folder and open its data fork. Call doWriteData to write the data to the data fork of the temporary file. Close the temporary file and the existing file. Swap the data in the existing and temporary files. Delete the temporary file, re-open the existing file. Call doCopyAppNameResource to copy the missing application name string to the resource fork.
Get data address and size. Set file mark at start. Write data to file. Adjust file size. Find the volume the file is on and flush the volume. Change window's document structure to flag window as "not touched".
doCopyAppName Resource
Create and open the file's resource fork. Call doCopyResource, specifying the application's missing application name 'STR' resource as the resource to copy. Close the resource fork.
Set source resource file, get resource, detach resource, set destination resource file and write resource to destination.
Note: On Mac OS X, when more than one document with unsaved changes is open and the user chooses the Quit command or a Quit Application Apple event is received, a Review Changes alert should be presented. doCloseCommand should be called if the user clicks the Review Changes... button in that alert.

Preliminaries-Creating a Document Structure

The contents of document files are displayed in windows. Ordinarily, your application should define a **document structure** that contains information about the window and information about the file whose contents are displayed in the window. The following is an example of a document structure for an application that handles text files:

```
typedef struct
{
  ControlHandle vScrollBarHdl;  // Handle to vertical scroll bar.
  ControlHandle hScrollBarHdl;  // Handle to horizontal scroll bar.
  SInt16        fileRefNum;     // File reference number for window's file.
  FSSpec        fileFSSpec;     // File's file system specification structure.
```

```
  TEHandle      textEditHdl;    // Handle to TextEdit structure.
  Boolean       windowTouched;  // Has window's data changed?
}   documentStructure;

typedef documentStructure *documentStructurePtr;
typedef documentStructure **documentStructureHdl;
```

Note the `fileRefNum` and `fileFSSpec` fields. Note also that the last field (`windowTouched`) is used to record whether the content of the document in memory differs from that in the associated file. Your application should set this field to `false` when it first reads in the file and immediately after each save, and to `true` when the content of the document in memory is changed by the user after the first read-in and after the subsequent saves. If the `windowTouched` field is set to `true` and the user attempts to close the document window, your application should present an alert asking the user whether the changed version of the document should be saved.

Document structures can be associated with the relevant window by storing a handle to the structure in the window object using the function `SetWRefCon`.

Creating a New Document Window

The user creates a new untitled document window using the New command in the File menu. In addition, it is usual for an application to open a new untitled document window when it receives an Open Application Apple event from the Finder. (See `doNewCommand` in Fig 4.)

Although the function that responds to the user choosing the New command and Open Application Apple event opens a new window, it should not create a new file. The reason for this is that, in the event, the user may elect not to save the document. It is thus preferable to wait until the user decides to save the new document before creating a file. Accordingly, the `fileRefNum` field of the new window's document structure should be set to `0` to indicate that no file is currently associated with this window.

Opening a File and Reading in Data

Your application will need to open a file when the user chooses the Open command from the File menu (see `doOpenCommand` in Fig 4) and when it receives Open Documents and (on Mac OS 8/9) Print Documents Apple events.

Opening the Navigation Services Open Dialog

Your application's initial response to the user choosing the Open command from the File menu should be to elicit a file selection from the user by creating and displaying a Navigation Services Open dialog (see Fig 6).

Figure 6. NAVIGATION SERVICES OPEN DIALOG





Calls to the Navigation Services 3.0 functions `NavCreateGetFileDialog` and `NavDialogRun` create and display the Navigation Services Open dialog. When the user addresses the dialog, selects one or more files, and clicks the Open button, your application examines the `selection` field of a `NavReplyRecord` structure (see Navigation Services, later in this chapter) and disposes of the dialog. The `selection` field is an Apple Event descriptor list (`AEDescList`). You can determine the number of files in the list by calling the Apple Event Manager function `AECountItems`. Each selected file object is described in an `AEDesc` structure. You can coerce this descriptor into a file system specification (`FSSpec`) structure to perform operations such as opening the file.

Creating a Window and Opening the File

The next steps are to call a function (`doNewDocWindow` in Fig 4) to create a window and associated document structure and open the selected file's data fork (`doOpenFile` in Fig 4).

The file's data fork is opened using FSpOpenDF:

```
OSErr  FSpOpenDF(spec,permission,refNum);
FSSpec *spec;       File system specification structure.
SInt8  permission;  Access mode.
short  *refNum;     Returned file reference number.
```

FSpOpenDF takes the FSSpec returned by Navigation Services as its first parameter. The permission field specifies the **access mode** for opening the file. The access mode may be specified using one of the following constants:

Constant	*Value*	*Description*
fsCurPerm	0	Whatever permission is allowed.
fsRdPerm	1	Read permission.
fsWrPerm	2	Write permission.
fsRdWrPerm	3	Exclusive read/write permission.
fsRdWrShPerm	4	Shared read/write permission.

FSpOpenDF returns, in its third parameter, a file reference number. This reference number should be saved to the window's document structure so that it can be readily retrieved for use as a parameter in calls to functions that read from and write to the file.

Reading File Data

When you have opened a file, you can read data from it. Ordinarily, you will want to read data from the file when the user first opens it. And your application will have to read data from the file when the user chooses the Revert command in the File menu to revert to the last saved version of the document (see doRevertCommand in Fig 4). Typically, a function for reading file data

- Retrieves the file reference number from the document structure.
- Calls SetFPos to set the file mark to the beginning of the file:

  ```
  OSErr  SetFPos(refNum,posMode,posOff);
  short refNum;   File reference number.
  short posMode;  Positioning mode.
  long  posOff;   Positioning offset.
  ```

  The posMode parameter must contain one of the following constants:

Constant	*Value*	*Description*
fsAtMark	0	Remain at current mark.
fsFromStart	1	Set mark relative to beginning of file.
fsFromLEOF	2	Set mark relative to logical end of file.
fsFromMark	3	Set mark relative to current mark.
rdVerify	64	Add to above for read-verify.

- Determines the number of bytes in the file by calling GetEOF:

  ```
  OSErr  GetEOF(refNum,logEOF);
  short refNum;   File reference number.
  long  *logEOF;  Receives length of file, in bytes.
  ```

- Calls FSRead to read the specified number of bytes from the file into the specified buffer:

  ```
  OSErr  FSRead(refNum,count,buffPtr);
  short refNum;    File reference number.
  long  *count;    On input: bytes to read. On output: actual bytes read.
  void  *buffPtr;  Address of buffer into which bytes are to be read.
  ```

Note that `FSRead` returns, in the `count` parameter, the actual number of bytes read.

Saving a File

The user can indicate that the current contents of a document should be saved:

- By choosing Save or Save As from the File menu.
- By clicking the Save button in the Navigation Services Save Changes alert when the user attempts to close a "touched" window.
- By clicking the Save button in the Navigation Services Save Changes alert when the user attempts to quit the application while a "touched" window remains open.

Handling the Save Command

To handle the Save command (see `doSaveCommand` in Fig 4), your application should

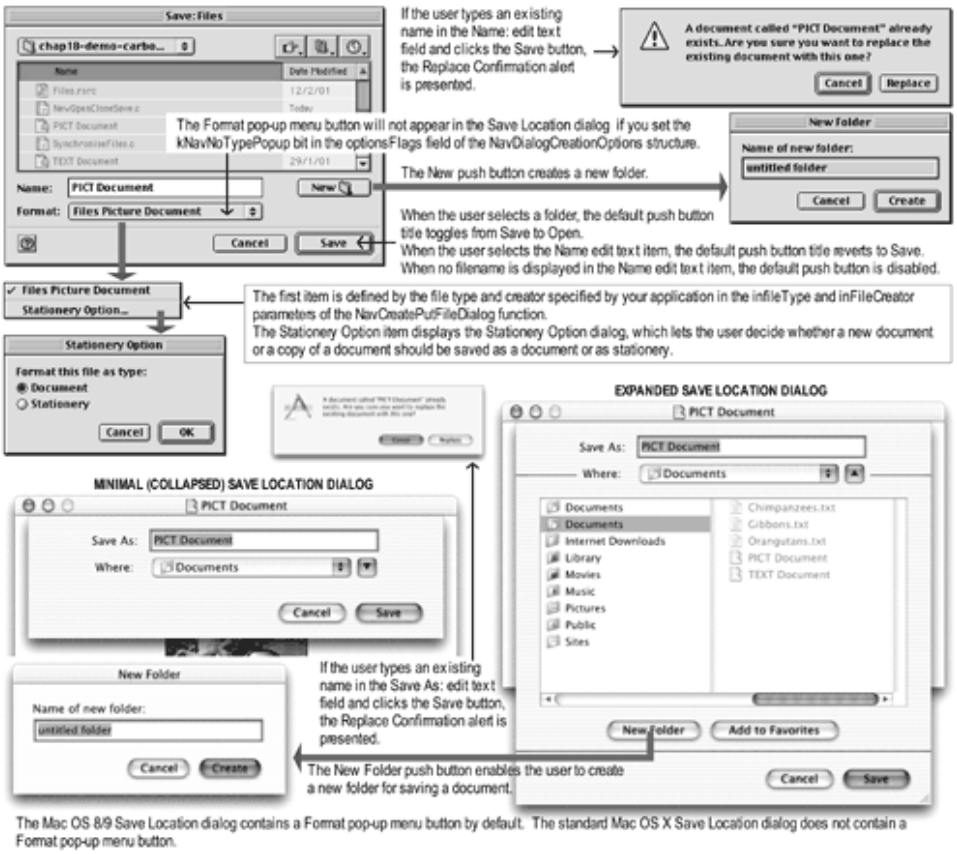
- Check the file reference number field of the window's document structure to determine if the window already has a file.
- If the window already has a file, call the function for writing files to disk (see `doWriteFile` in Fig 4). If the window does not have a file, call the function for handling the Save As command.

Handling the Save As Command

To handle the Save As… command (see `doSaveAsCommand` in Fig 4), your application should proceed as follows

- Call the Navigation Services 3.0 functions `NavCreatePutFileDialog` and `NavDialogRun` to create and display the Navigation Services Save Location dialog (see Fig 10).

Figure 10. THE SAVE LOCATION DIALOG BOX (PARTIAL) AND ASSOCIATED DIALOGS AND ALERTS


Save: Files
chap18-demo-carbo...
Name:
PICT Document
Format:
Files Picture Document
New
Cancel
Save
If the user types an existing name in the Name: edit text field and clicks the Save button, the Replace Confirmation alert is presented.
A document called "PICT Document" already exists. Are you sure you want to replace the existing document with this one?
Cancel
Replace
The Format pop-up menu button will not appear in the Save Location dialog if you set the kNavNoTypePopup bit in the optionsFlags field of the NavDialogCreationOptions structure.
The New push button creates a new folder.
New Folder
Name of new folder:
untitled folder
Cancel
Create
When the user selects a folder, the default push button title toggles from Save to Open.
When the user selects the Name edit text item, the default push button title reverts to Save.
When no filename is displayed in the Name edit text item, the default push button is disabled.
Files Picture Document
Stationery Option...
The first item is defined by the file type and creator specified by your application in the inFileType and inFileCreator parameters of the NavCreatePutFileDialog function.
The Stationery Option item displays the Stationery Option dialog, which lets the user decide whether a new document or a copy of a document should be saved as a document or as stationery.
Stationery Option
Format this file as type:
Document
Stationery
Cancel
OK
EXPANDED SAVE LOCATION DIALOG
PICT Document
Save As:
PICT Document
Where:
Documents
Documents
Internet Downloads
Library
Movies
Music
Pictures
Public
Sites
Chimpanzees.txt
Gibbons.txt
Orangutans.txt
PICT Document
TEXT Document
New Folder
Add to Favorites
Cancel
Save
MINIMAL (COLLAPSED) SAVE LOCATION DIALOG
PICT Document
Save As:
PICT Document
Where:
Documents
Cancel
Save
New Folder
Name of new folder:
untitled folder
Cancel
Create
If the user types an existing name in the Save As: edit text field and clicks the Save button, the Replace Confirmation alert is presented.
The New Folder push button enables the user to create a new folder for saving a document.
The Mac OS 8/9 Save Location dialog contains a Format pop-up menu button by default. The standard Mac OS X Save Location dialog does not contain a Format pop-up menu button.

When the user addresses the dialog and clicks the Save button, your application examines the `selection` field of a `NavReplyRecord` structure (see Navigation Services, below) and disposes of the dialog. The `selection` field is an Apple

Event descriptor list (`AEDescList`). The file object is described in an `AEDesc` structure. If your application is running on Mac OS X, you will be able to coerce this data to type `FSRef`. If this coercion fails (meaning that your application is running on Mac OS 8/9) you will be able to coerce the data to type `FSSpec`. The `FSRef` or `FSSpec` will be required for the save operation.

- ***Save Using FSRef.*** If the coercion to type `FSRef` succeeds

    - Call `AEGetDescData` to extract the data from the `dataHandle` field of the `AEDesc` structure. This is the `FSRef` for the parent directory.

    - Call `CFStringGetCharacters` to extract into a buffer the contents of the string referenced in the `saveFileName` field of the `NavReplyRecord` structure.

    - If the `replacing` field of the `NavReplyRecord` structure contains `true`, call the HFS Plus API function `FSMakeFSRefUnicode` to create an `FSRef` for the file, passing in the `FSRef` for the parent directory and the extracted filename characters. Pass this `FSRef` in a call to `FSDeleteObject` to delete the file:

      ```
      OSErr  FSDeleteObject(ref);
      const FSRef *ref;  Pointer to FSRef specifying file or directory to delete.
      ```

    - Call `FSCreateFileUnicode`, passing in the `FSRef` for the parent directory and the extracted filename characters, to create a new file with a Unicode name:

      ```
      OSErr  FSCreateFileUnicode(parentRef,nameLength,name,whichInfo,catalogInfo,
                                 newRef,newSpec);
      const FSRef         *parentRef;    FSRef for directory where file to be created.
      UniCharCount        nameLength;    Length of file's name.
      const UniChar       *name;         Unicode name for file.
      FSCatalogInfoBitmap whichInfo;     Catalog information fields to be set, if any.
      const FSCatalogInfo *catalogInfo;  Values of file's catalog infoformation.
      FSRef               *newRef;       On return, FSRef for new file.
      FSSpec              *newSpec;      On return, FSSpec for new file.
      ```

    - Call `FSpGetFInfo`, passing in the `FSSpec` received in the last parameter of the call to `FSCreateFileUnicode`. Assign the file type and creator to the relevant fields of the obtained `Finfo` structure and call `FSpSetFInfo` to set the Finder information.

    - Assign the file system specification (`FSSpec`) structure to the file system specification structure field of the window's document structure.

    - Call `FSpOpenDF` to open the data fork.

    - Assign the file reference number returned by `FSpOpenDF` to the file reference number field of the window's document structure.

    - Call `SetWTitle` to set the window's title, using the string extracted from the `name` field of the file system specification (`FSSpec`) structure.

    - Call the function for writing files to disk (see `doWriteFile` in Fig 4).

- ***Save Using FSSpec.*** If the coercion to type `FSRef` does not succeed

    - Call the Navigation Services 3.0 function `NavDialogGetSaveFileName` to get the file name from the edit text field of the Save Location dialog, convert it to a Pascal string using `CFStringGetPascalString`, and assign that string to the name field of the file system specification (`FSSpec`) structure.

    - If the `replacing` field of a `NavReplyRecord` structure does not contain `true`, call `FSpCreate` to create a new file and set the file type and creator:

      ```
      OSErr  FSpCreate(spec,creator,fileType,sciptTag);
      FSSpec     *spec;      File system specification structure.
      OSType     creator;    File creator.
      OSType     fileType;   File type.
      ScriptCode scriptTag;  Code of script system in which filename is displayed.
      ```

- Assign the coerced file system specification (`FSSpec`) structure to the file system specification structure field of the window's document structure.
- If the window already has a file (that is, if the file reference number field of the document structure does not contain `0`), close that file with a call to `FSClose`

```
OSErr  FSClose(refNum);
short refNum;  File reference number.
```

- Call `FSpOpenDF` to open the data fork.
- Assign the file reference number returned by `FSpOpenDF` to the file reference number field of the window's document structure.
- Call `SetWTitle` to set the window's title, using the string extracted from the `name` field of the file system specification (`FSSpec`) structure.
- Call the function for writing files to disk (see `doWriteFile` in Fig 4).

Writing File Data

The function for writing data (see `doWriteFile` in Fig 4) should write to a temporary file, not to the document file. If you write directly to the document's file, you risk corrupting that file if the write operation does not complete successfully. For saving data *safely* to disk, write the data to a temporary file and then, assuming the temporary file has been written successfully, swap the contents of the temporary file and the document's file.

The procedure for updating a file safely is as follows:

- Get the file system specification from the document structure.
- Create a temporary filename for the temporary file.
- Call `FindFolder` to find the temporary folder on the file's volume, or create it if necessary:

```
OSErr  FindFolder(vRefNum,folderType,createFolder,foundVRefNum,foundDirID);
short   vRefNum;        Volume reference number.
OSType  folderType;     Folder type for volume.
Boolean createFolder;   kCreateFolder or kDontCreateFolder.
short   *foundVRefNum;  Volume reference number for folder found.
long    *foundDirID;    Directory ID of folder found.
```

- Call `FSMakeFSSpec` to make a file system specification structure for the temporary file:

```
OSErr  FSMakeFSSpec(vRefNum,dirID,fileName,spec);
short           vRefNum;   Volume reference number.
long            dirID;     Parent directory ID.
ConstStr255Param fileName;  Full or partial pathname.
FSSpec          spec;      Pointer to FSSpec structure.
```

- Call `FSpCreate` to create the temporary file, and `FSpOpenDF` to open the temporary file's data fork.
- Call the function for writing data to a file (see `doWriteData` in Fig 4). This function should:
  - Retrieve the address and length of the buffer (for example, from a TextEdit structure).
  - Call `SetFPos` to set the file mark to the beginning of the file.
  - Call `FSWrite` to write the buffer to the file

```
OSErr  FSWrite(refNum,count,buffPtr);
short      refNum;    File reference number.
```

```
long       *count;    On input: bytes to write. On output: bytes written.
const void *buffPtr;  Address of buffer containing data to write.
```

- Call `SetEOF` to resize the file to the number of bytes actually written

```
OSErr  SetEOF(refNum,logEOF);
short refNum;  File reference number.
long  logEOF;  Logical end-of-file.
```

- Call `GetVRefNum` to determine the volume containing the file

```
OSErr  GetVRefNum(refNum,vRefNum);
short refNum;    File reference number.
short *vRefNum;  Receives volume reference number.
```

- Call `FlushVol` to flush the volume

```
OSErr  FlushVol(volName,vRefNum);
ConstStr63Param volName;  Pointer to name of mounted volume
short           vRefNum;  Volume reference number.
```

Flushing the volume ensures that both the file's data and the file's catalog entry are updated.

Note

The catalog entry for a file contains fields that describe the physical data (such as the first allocation block and the physical and logical ends of both the resource and data forks) and fields that describe the file within the file system, such as file ID and parent directory ID.

- Call `FSClose` to close the temporary file.
- Call `FSClose` to close the existing file.
- Call `FSpExchangeFiles` to swap the contents of the temporary file and the existing file

```
OSErr  FSpExchangeFiles(source,dest);
const FSSpec *source;  Source file.
const FSSpec *dest;    Destination file.
```

`FSpExchangeFiles` does not actually move the data on the volume. It merely changes the information in the volume's catalog file and, if the files are open, their file control blocks (FCBs).

- Call `FSpDelete` to delete the temporary file

```
OSErr  FSpDelete(spec);
const FSSpec *spec;  File system specification.
```

- Call `FSpOpenDF` to re-open the data fork of the existing file.

As a final step for Mac OS 8/9, you should call a function that copies the missing application name string resource (see Chapter 9) from the resource fork of the application file to the resource fork of the newly created file. This function (`doCopyAppNameResource` in Fig 4) should

- Call `FSpCreateResFile` to create the new file's resource fork

```
void  FSpCreateResFile(spec,creator,fileType,sciptTag);
const FSSpec *spec;      File system specification structure.
OSType       creator;    File creator.
OSType       fileType;   File type.
ScriptCode   scriptTag;  Code of script system.
```

- Call `FSpOpenResFile` to open the resource fork

```
short  FSpOpenResFile(spec,permission);
const FSSpec *spec;       File system specification structure.
SignedByte   permission;  Permission code.
```

  The constants used to specify the access mode in the `FSpOpenDF` call (see above) are also used to specify the permission code in the `FSpOpenResFile` call.

- Call a function (`doCopyResource` in Fig 4), which copies specified resources from one resource file to another, to copy the missing-application name `'STR '` resource (ID -16396) from your application's resource fork to the resource fork of the newly-created file.

- Call `FSClose` to close the resource fork.

Reverting to a Saved File

To allow the user to revert to the last saved version of a document, your application can include a Revert command in the File menu. To handle this command (see `doRevertCommand` in Fig 4), you should call the Navigation Services 3.0 functions `NavCreateAskDiscardChangesDialog` and `NavDialogRun` to create and display a Navigation Services Discard Changes alert (see Fig 13). When the user addresses the dialog, and clicks on the OK button, you simply call your function for reading file data (`doReadFile` in Fig 4) to read the file back into the window.

Figure 13. DISCARD CHANGES ALERT



Closing a File

Your application should ordinarily close a file when the user clicks in the close box of the associated window or chooses the Close command from the File menu. You may also need to close files when the user chooses Quit from the File menu or a Quit Application Apple event is received from the Finder.

When your application needs to close a file, it should first (c)check whether the associated window has been "touched" (see `doCloseCommand` in Fig 4). If the window has been "touched," you should call the Navigation Services 3.0 functions `NavCreateAskSaveChangesDialog` and `NavDialogRun` to create and display a Navigation Services Save Changes alert (see Fig 12). When the user addresses the dialog

- If the Save button is clicked, call the function for saving files (`doSaveCommand` in Fig 4), call `FSClose` to close the file, call `FlushVol` to ensure that both the file's data and the file's catalog entry are updated, set the file reference number field in the document structure to `0`, and release memory associated with the storage of the file's data. Then dispose of the document structure and, finally, the window.

- If the `Don't Save` button is clicked, perform the same actions as are performed when the `Save` button is clicked except for the call to the function for saving files.

Figure 12. SAVE CHANGES ALERTS (CLOSING DOCUMENT AND QUITTING APPLICATION)


Do you want to save changes to the files document "PICT Document" before closing it?
Don't Save
Cancel
Save
PICT Document
Save changes before closing
Do you want to save the changes you made to the document PICT Document"?
Don't Save
Cancel
Save
Do you want to save changes to the document "PICT Document" before quitting this application?
Don't Save
Cancel
Save
PICT Document
Save changes before quitting
Do you want to save the changes you made to the document PICT Document"?
Don't Save
Cancel
Save

If the window has not been "touched," perform the same actions as are performed when the Save button is clicked in a Save Changes alert except for the call to the function for saving files.

File Synchronization Functions

It is possible that, while a document file is open, the user may drag its Finder icon or proxy icon to another folder (including the Trash) or change the name of the file via the Finder icon. The application has no way of knowing that this has happened and will assume, unless it is informed otherwise, that the document's file is still at its original location with its original name. For this reason, applications often include a frequently called **file synchronization function** that synchronizes the application with the actual current location (and name) of its currently open document files.

In applications that use the Classic event model, file synchronization functions should be called after every call to `WaitNextEvent`. In applications that use the Carbon event model, a timer should be installed to trigger repeated calls to the file synchronization function. For each of the application's document windows, the synchronization function should update the application's internal data structures to match that of the document file as it exists on disk. The function should also ensure that, where necessary, the name of the document window is changed to match the current name of the document file on disk and close the document window if the document file has been moved to the Trash folder.

Navigation Services

The user interface for opening and saving files, confirming saves and discarding changes, choosing a volume, folder, file, or file object, creating a new folder, file format translation, and easy navigation of the file system is provided by Navigation Services.

The following reflects Navigation Service 3.0, which was introduced with CarbonLib 1.1, and which established a new model for the creation, display, and handling of Navigation Services dialogs and alerts. Navigation Services 3.0 also introduced support for Unicode and, on Mac OS X, support for sheets and the ability to specify the modality of a dialog.

Navigation Services Dialogs and Alerts

The primary dialogs created by Navigation Services are as follows:

- Open.
- Save Location.
- Choose a Volume.
- Choose a Folder.
- Choose a File.
- Choose a File Object.

The primary alerts created by Navigation Services are as follows:

- Save Changes
- Discard Changes.

A further alert, which is applicable only on Mac OS X, and for which no Navigation Services creation function existed at the time of the first release of Mac OS X, is the Review Changes alert.

The secondary dialogs and alerts created by Navigation Services are as follows:

- New Folder dialog.
- Replace Confirmation alert.
- Mac OS 8/9 Stationery option dialog.

Modality

On Mac OS 8/9, all primary Navigation Services dialogs are movable modal provided an application-defined event handling (callback) function is provided.

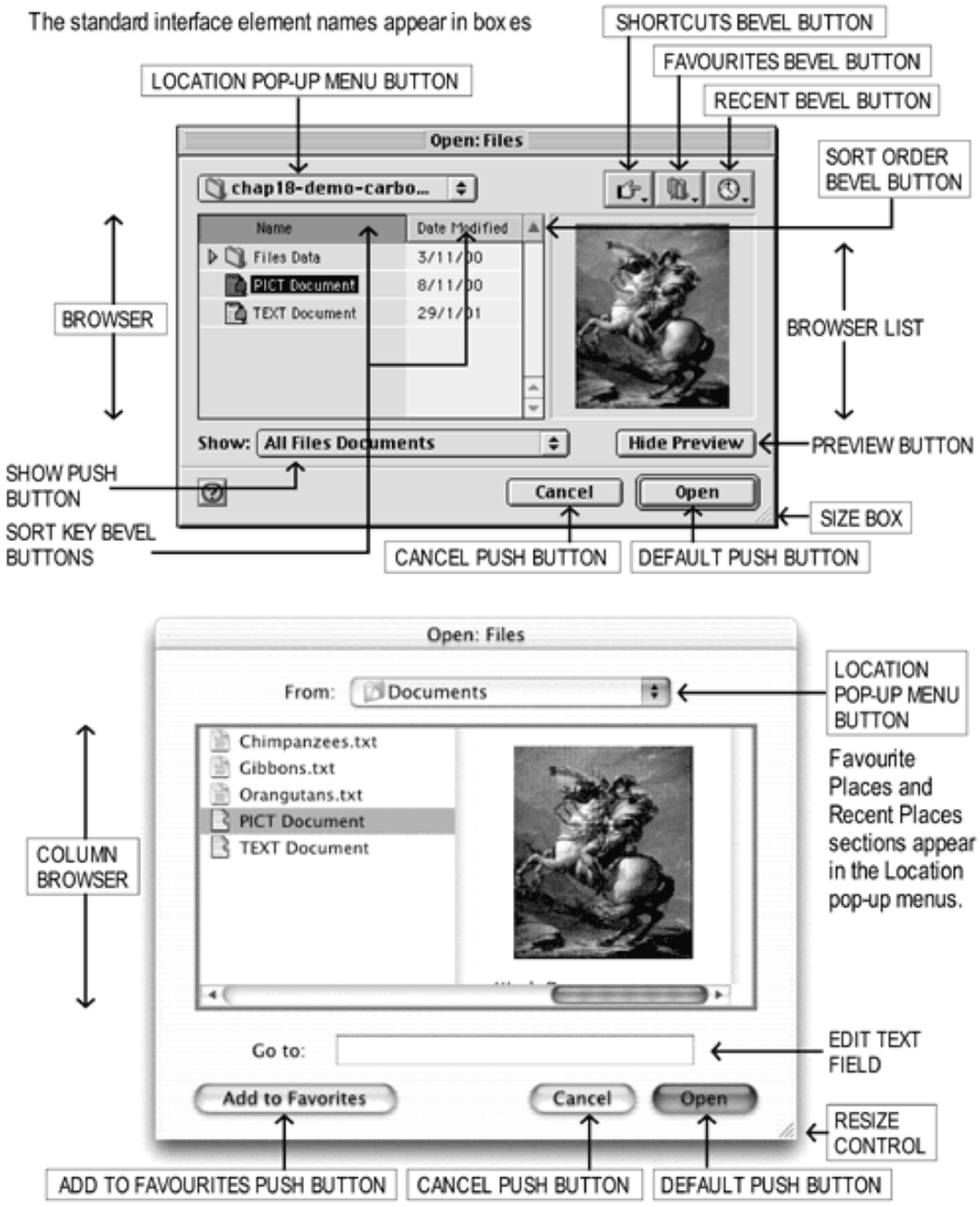
On Mac OS X, your application should ensure that:

- The Save Location dialog, Save Changes alert, and Discard Changes alert are window-modal (that is, sheets).
- The other primary dialogs are application-modal.

Standard User Interface Elements in Primary Dialogs

The standard user interface elements in Navigation Services primary dialogs are shown in Fig 5.

Figure 5. STANDARD USER INERFACE ELEMENTS IN NAVIGATION SERVICES DIALOGS



Preview

On Mac OS 8/9, the user can toggle on and off a preview area using the Show/Hide Preview push button in the Open dialog. A preview of any file that contains a valid `'pnot'` resource will be displayed in this area.

On Mac OS X, the preview appears in the column browser as shown in Fig 5. For files of type 'TEXT' a preview is automatically created.

Persistence

Navigation Services has the capability to store information, and to store it on a per-application basis. This ability is called **persistence.** For example, when a primary dialog is displayed, the browser defaults to the directory location that was in use when that particular dialog was last closed by that particular application. In addition, if a file or folder was selected when the dialog was last closed, that file or folder is automatically selected when the dialog is re-opened. For dialogs that are not sheets, the size, position, and, on Mac OS 8/9, sort key and sort order are also remembered for each application.

Creating and Displaying an Open Dialog

The Open dialog (see Fig 6) is created by a call to `NavCreateGetFileDialog` and displayed by a call to `NavDialogRun`. You pass a universal procedure pointer to an application-defined event handling (callback) function in the `inEventProc` parameter of `NavCreateGetFileDialog`.

The NavDialogCreationOptions Structure

You pass a pointer to a `NavDialogCreationOptions` structure, which specifies options controlling the appearance and behavior of the dialog, in the `inOptions` parameter of `NavCreateGetFileDialog`. The `NavDialogCreationOptions` structure is as follows:

```
struct NavDialogCreationOptions
{
  UInt16               version;
  NavDialogOptionFlags optionFlags;
  Point                location;
  CFStringRef          clientName;
  CFStringRef          windowTitle;
  CFStringRef          actionButtonLabel;
  CFStringRef          cancelButtonLabel;
  CFStringRef          saveFileName;
  CFStringRef          message;
  UInt32               preferenceKey;
  CFArrayRef           popupExtension;
  WindowModality       modality;
  WindowRef            parentWindow;
  char                 reserved[16];
} ;
typedef struct NavDialogCreationOptions NavDialogCreationOptions;
```

Field Descriptions

`optionsFlags` One of the following constants of type `NavDialogOptionFlags`:

Constant	*Description*

`kNavDefaultNavDlogOptions`	Use default options. Sets the following bits:
	
	`kNavDontAddTranslateItems`
	`kNavAllowStationery`
	`kNavAllowPreviews`
	`kNavAllowMultipleFiles`

`kNavNoTypePopup`	Don't show file type pop-up menu button.
`kNavDontAutoTranslate`	Don't auto-translate files. (Application will translate.)

`kNavDontAddTranslateItems`	Don't add translation options in Show pop-up menu.
`kNavAllFilesInPopup`	Add `All Documents` menu item in file type pop-up.
`kNavAllowStationery`	Allow stationery files.
`kNavAllowPreviews`	Allow preview to show.
`kNavAllowMultipleFiles`	Allow multiple items to be selected.
`kNavAllowInvisibleFiles`	Allow invisible items to be shown.
`kNavDontResolveAliases`	Don't resolve aliases.
`kNavSelectDefaultLocation`	Make the default location the browser selection.
`kNavSelectAllReadableItem`	Make dialog select `All Readable Documents` on open.
`kNavSupportPackages`	Recognize file system packages.
`kNavAllowOpenPackages`	Allow opening of packages.
`kNavDontAddRecents`	Don't add chosen objects to Recents list.
`kNavDontUseCustomFrame`	Don't draw the custom area bevel frame.
`kNavDontConfirmReplacement`	Don't show the "Replace File?" alert on save conflict.

`location` Location of the upper-left of the dialog (global coordinates). The dialog will appear at the location at which it was last closed if the `optionFlags` field is assigned `NULL` or this field is assigned (`-1,-1`).

`clientName` Name of your application. This will appear in the dialog's title bar.

`windowTitle` Overrides the default window title.

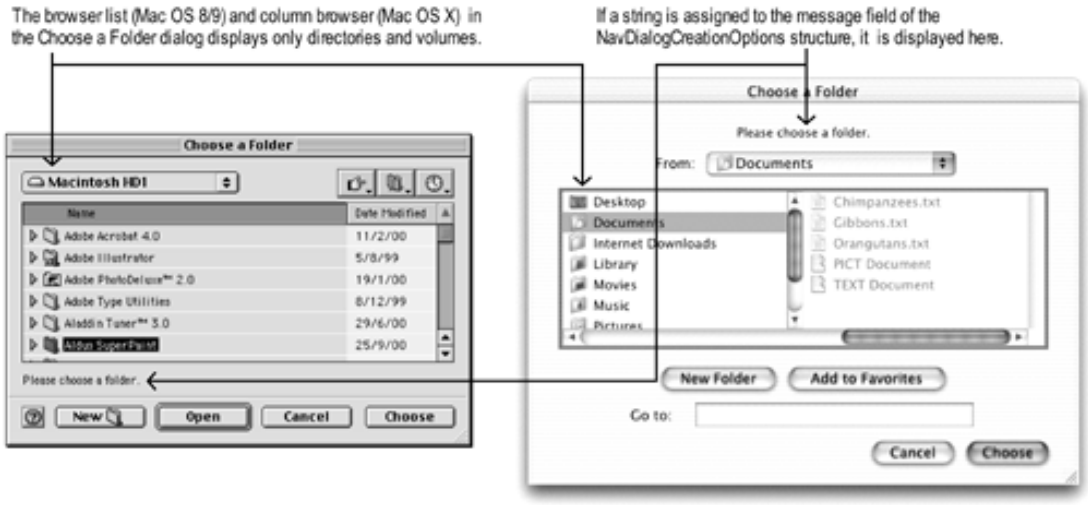
`actionButtonLabel` Title for the dialog's OK push button. If a title is not assigned, the push button will use the default title (Open or Save).

`cancelButtonLabel` Title for the dialog's Cancel push button. If a title is not assigned, the push button will use the default title (Cancel).

`savedFileName` Default file name for a saved file.

`message` A string, which is displayed in the dialog, providing additional instructions to the user. (For an example, see Fig 11).

Figure 11. CHOOSE A FOLDER DIALOG



`preferenceKey` A value that identifies the set of dialog preferences that should be used. Assign `0` if you do not wish to provide a preference key.

`popupExtension` A handle to one or more structures of type `NavMenuItemSpec` used to add extra menu items to an Open dialog's Show pop-up menu or a Save Location dialog's Format pop-up menu.

`modality` The dialog's modality (relevant on Mac OS X only). Relevant Window Manager constants are:

```
kWindowModalityNone
kWindowModalitySystemModal
kWindowModalityAppModal
kWindowModalityWindowModal
```

`parentWindow` The dialog's parent window. (Relevant on Mac OS X only when the dialog is window-modal.)

The function `NavGetDefaultDialogCreationOptions` may be called to initialise a structure of type `NavDialogCreationOptions` with the default dialog options, which are as follows:

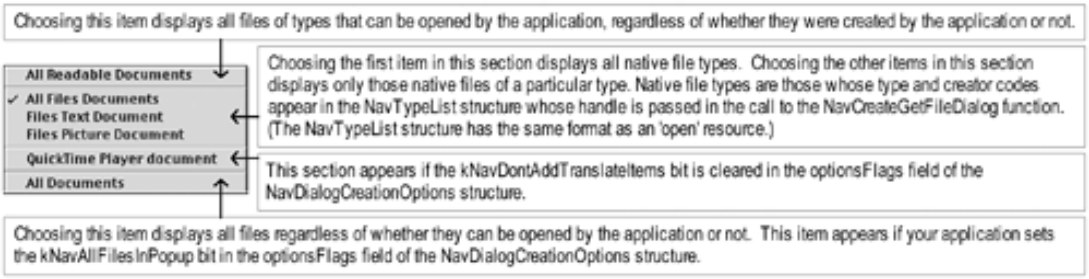
- `Show` and `Format` pop-up menu buttons are displayed in the Open and Save Location dialogs (Mac OS 8/9).
- Files are auto-translated. (This implies that the application will not translate.)
- Translation options are not included in the `Show` pop-up menu in the Open dialog.
- The `All Documents` item is not included in the `Show` pop-up menu in the Open dialog.
- The `Stationery Option` item is included in the `Format` pop-up menu in the Save Location dialog.
- Previews of selected files, when available, are displayed in the Open dialog.
- Selection of multiple files in the browser list/column browser in the Open dialog is allowed.
- Invisible files are nor displayed.
- Aliases are not resolved.
- The default location in the browser list/column browser is not selected.
- The `All Readable Documents` is not made the default selection in the `Show` pop-up menu in the Open dialog.
- File system packages are not displayed.
- File system packages cannot be opened and navigated.
- Chosen file objects are added to the Recents list.
- A border is drawn around the custom area.
- The default titles for the OK and Cancel buttons are used.
- No message is displayed in the dialog.

The Show Pop-up Menu

The types of files to be displayed in the browser list may be chosen from a list of available **file types** in the Show pop-up

menu in the Open dialog (see Fig 7). This list is built from information supplied by your application in a structure of type `NavTypeList` (see below), a handle to which you pass in the `inTypeList` parameter of `NavCreateGetFileDialog`.

Figure 7. THE SHOW POP-UP MENU AND FILE TYPE OPTIONS (MAC OS 8/9)


Choosing this item displays all files of types that can be opened by the application, regardless of whether they were created by the application or not.
All Readable Documents
All Files Documents
Files Text Document
Files Picture Document
QuickTime Player document
All Documents
Choosing the first item in this section displays all native file types. Choosing the other items in this section displays only those native files of a particular type. Native file types are those whose type and creator codes appear in the NavTypeList structure whose handle is passed in the call to the NavCreateGetFileDialog function. (The NavTypeList structure has the same format as an 'open' resource.)
This section appears if the kNavDontAddTranslateItems bit is cleared in the optionsFlags field of the NavDialogCreationOptions structure.
Choosing this item displays all files regardless of whether they can be opened by the application or not. This item appears if your application sets the kNavAllFilesInPopup bit in the optionsFlags field of the NavDialogCreationOptions structure.

The Show pop-up menu button will not appear in the Open dialog if you pass `NULL` in the `inTypeList` parameter of the `NavCreateGetFileDialog` function or if you set the `kNavNoTypePopup` bit in the `optionsFlags` field of the `NavDialogCreationOptions` structure passed in the call to `NavCreateGetFileDialog`.

If a handle to a `NavTypeList` structure is passed in the `inTypeList` parameter and the `kNavNoTypePopup` bit is set

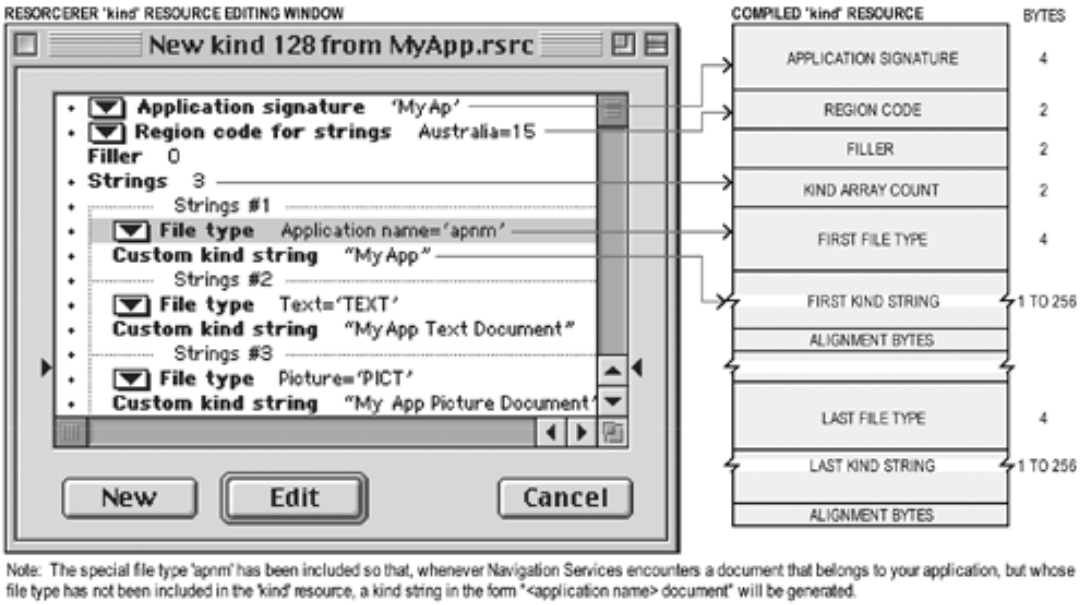
- All items in the browser will be deactivated except for the file types specified in the `NavTypeList` structure whether they were created by the application or not.
- The `Show` pop-up menu button will not appear.

Native File Types Section

The first item in the **native file types** section of the Mac OS 8/9 Show pop-up menu defaults to All Known Documents if you do not assign the name of your application to the `clientName` field of the `NavDialogCreationOptions` structure passed in the `dialogOptions` parameter of the `NavCreateGetFileDialog` function.

The remaining items in the native file types section will default to <Application Name> Document unless you provide **kind strings** to describe the file types included in your `NavTypeList` structure. For Mac OS 8/9, you can do this by including a **kind resource** (a resource of type `'kind'`) in your application's resource fork. Fig 8 shows the structure of a compiled `'kind'` resource and such a resource being created using Resorcerer.

Figure 8. STRUCTURE A COMPILED "kind" RESOURCE AND CREATING A 'kind' RESOURCE USING RESOURCERER



Note

The kind strings from your application's `'kind'` resource also appear in the Kind column in Finder window list views.

For Mac OS X, the `'kind'` resource is ignored if you provide necessary information in your application's `'plst'` resource. The relevant keys are `CFBundleDevelopmentRegion`, `CFBundleSignature`, and `CFBundleDocumentTypes`. `'apnm'` as a `CFBundleTypeOSTypes` has same effect as in `'kind'` resource.

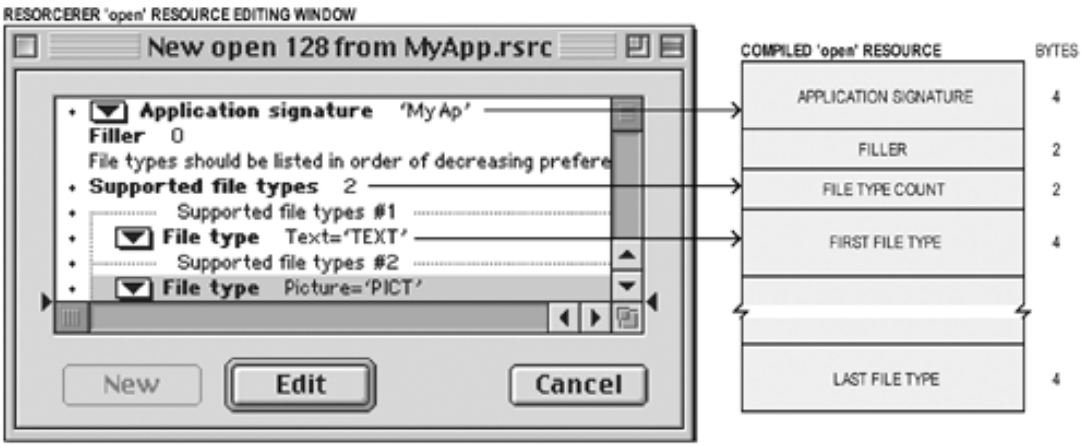
The NavTypeList Structure

The `NavTypeList` structure, which defines the list of file types that your application is capable of opening, is as follows:

```
struct NavTypeList
{
  OSType componentSignature;  // Your application signature.
  short  reserved;
  short  osTypeCount;         // How many file types will be defined.
  OSType osType[1];           // A list of file types your application can open.
} ;
typedef struct NavTypeList NavTypeList;
typedef NavTypeList *NavTypeListPtr;
typedef NavTypeListPtr *NavTypeListHandle;
```

You can create your file type list dynamically or you can use an `'open'` resource. Fig 9 shows the structure of a compiled `'open'` resource and such a resource being created using Resorcerer.

Figure 9. STRUCTURE OF A COMPILED 'open' RESOURCE AND CREATING AN 'open' RESOURCE USING RESOURCERER



Creating and Displaying a Save Location Dialog

The Save Location dialog (see Fig 10) is created by a call to `NavCreatePutFileDialog` and displayed by a call to `NavDialogRun`. You pass a universal procedure pointer to an application-defined event handling (callback) function in the `inEventProc` parameter of `NavCreatePutFileDialog`.

As with `NavCreateGetFileDialog`, you pass a pointer to a `NavDialogCreationOptions` structure in the `inOptions` parameter of `NavCreatePutFileDialog`. Other parameters allow you to specify file type and file creator.

Creating and Displaying a Choose a Folder Dialog

The Choose a Folder dialog (see Fig 11) is created by a call to `NavCreateChooseFolderDialog` and displayed by a call to `NavDialogRun`. You pass a universal procedure pointer to an application-defined event handling (callback) function in the `inEventProc` parameter of `NavCreateChooseFolderDialog` and a pointer to a `NavDialogCreationOptions` structure in the `inOptions` parameter.

The other dialogs in the Choose family are created and displayed in a similar manner:

- The Choose a Volume dialog is created by a call to `NavCreateChooseVolumeDialog`.
- The Choose a File dialog is created by a call to `NavCreateChooseFileDialog`, and may be used when you want the user to select a file for a purpose other than opening. The file could be, for example, a preferences file or a dictionary file.
- The Choose a File Object dialog is created by a call to `NavCreateChooseObjectDialog`, and may be used when you need the user to select an object that might be one of several different types.

Creating and Displaying Primary Alerts

Save Changes Alert

The Save Changes alert (see Fig 12) is created by a call to `NavCreateAskSaveChangesDialog` and displayed by a call to `NavDialogRun`. You pass a universal procedure pointer to an application-defined event handling (callback) function in the `inEventProc` parameter of `NavCreateAskSaveChangesDialog` and a pointer to a `NavDialogCreationOptions` structure in the `inOptions` parameter.

One of the following constants is passed in the `inAction` parameter of the `NavCreateAskSaveChangesDialog` function:

```
kNavSaveChangesClosingDocument      = 1
```

```
kNavSaveChangesQuittingApplication = 2
kNavSaveChangesOther               = 0
```

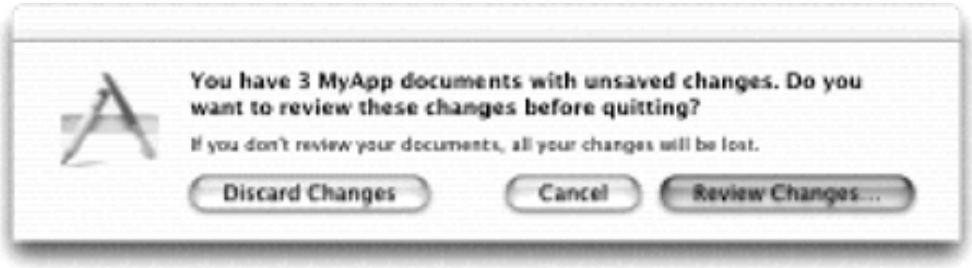
Discard Changes Alert

To support the Revert command in your application's File menu, Navigation Services provides the Discard Changes alert. The Discard Changes alert (see Fig 13) is created by a call to `NavCreateAskDiscardChangesDialog` and displayed by a call to `NavDialogRun`. You pass a universal procedure pointer to an application-defined event handling (callback) function in the `inEventProc` parameter of `NavCreateAskDiscardChangesDialog` and a pointer to a `NavDialogCreationOptions` structure in the `inOptions` parameter.

Review Changes Alert-Mac OS X

On Mac OS X, when the user attempts to quit your application when there is more than one document with unsaved changes open, your application should present a Review Changes alert (see Fig 14). No Navigation Services creation function existed at the time of writing; accordingly, at the time of writing, it was necessary to create, display, and handle this alert using `StandardAlert` or `CreateStandardAlert`.

Figure 14. REVIEW UNSAVED ALERT


You have 3 MyApp documents with unsaved changes. Do you want to review these changes before quitting?
If you don't review your documents, all your changes will be lost.
Discard Changes
Cancel
Review Changes...

A click on the Discard Changes button should cause all windows to close (without saving changes) and the application to close down. A click on the Cancel button should cancel the Quit command, keeping the application running. A click on the Review Changes button should cause each window with unsaved changes to be sequentially presented to the user with a Save Changes alert presented.

Event Handling in the Primary Dialogs

Event-Handling Function

As previously stated, you pass a universal procedure pointer to an application-defined event handling (callback) function in the `inEventProc` parameter of those functions that create the Navigation Services primary dialogs. For an event handling function named `myNavEventFunction`, you would declare the function as follows:

```
void  myNavEventFunction(NavEventCallbackMessage callBackSelector,
                         NavCBRecPtr callBackParms,NavCallBackUserData callBackUD)
```

`callBackSelector` The type of event, as represented by an event message constant. Typical event message constants and their meanings are as follows:

Constant	*Value*	*Description*
`kNavCBUserAction`	`12`	The user has taken an action such as clicking on an Open or Save button.
`kNavCBEvent`	`0`	An event has occurred. Receipt of this event type allows your handler to update other windows, track controls, etc.

kNavCBTerminate	3	The dialog is about to be closed.

callBackParms A pointer to a NavCBRec structure, which contains data used by your application to process the event:

```
struct NavCBRec
{
  UInt16        version;
  NavDialogRef  context;
  WindowRef     window;
  Rect          customRect;
  Rect          previewRect;
  NavEventData  eventData;
  NavUserAction userAction;
  char          reserved[218];
} ;
typedef struct NavCBRec NavCBRec;
typedef NavCBRec *NavCBRecPtr;
```

callBackUD A pointer to a value passed in the inClientData parameter of the dialog creation functions.

kNavCBUserAction Message Received

When the kNavCBUserAction message is received, your application typically calls NavDialogGetReply to obtain the results of the dialog session, which are returned in a NavReplyRecord structure.

The NavReplyRecord Structure

```
struct NavReplyRecord
{
  UInt16                        version;
  Boolean                       validRecord;
  Boolean                       replacing;
  Boolean                       isStationery;
  Boolean                       translationNeeded;
  AEDescList                    selection;
  ScriptCode                    keyScript;
  FileTranslationSpecArrayHandle fileTranslation;
  UInt32                        reserved1;
  CFStringRef                   saveFileName;
  char                          reserved[227];
} ;
typedef struct NavReplyRecord  NavReplyRecord;
```

Field Descriptions

validRecord true if the user closed a dialog by clicking the Open button in an Open dialog, the Save button in a Save Location dialog, or the Choose button in a Choose dialog, or by pressing the Return or Enter key. If this field is false, the remaining fields do not contain valid data.

replacing true if the user chooses to save a file by replacing an existing file.

isStationery true if the file about to be saved should be saved as a stationery document.

translationNeeded true if translation was or will be needed for selected files in Open and Save Location dialogs.

`selection` An Apple event descriptor list (`AEDescList`) created from `FSSpec` or `FSRef` references to selected items. You can determine the number of items in the list by calling `AECountItems`. Each selected item is described in an `AEDesc` structure by the descriptor type `typeFSS` or `typeFSRef`. This descriptor can be coerced into an `FSSpec` or `FSRef` preparatory to, for example, opening the file.

`keyScript` Keyboard script system used for the filename.

`fileTranslation` Handle to a `FileTranslationSpec` structure, which contains a corresponding translation array for each file reference returned in the `selection` field.

`saveFileName` The save file name in `CFString` form.

This field was introduced with Navigation Services 3.0 because there is no way to fit a 255-character `Unicode` name into the `name` field of an `FSSpec` or into an `FSRef`. (See `selection` field.)

Note 1: On Mac OS 9, you will never get a file name that won't fit into the `name` field of an `FSSpec` structure.

Note 2: On Mac OS X, you cannot reliably convert the name in the `saveFileName` field to a 31-byte Pascal string.

When your application has finished using this structure, it should dispose of it by calling the function `NavDisposeReply`.

Responding to User Actions

If the `validRecord` field of the `NavReplyRecord` structure contains true, your application typically calls `NavDialogGetUserAction` to determine the user action, as represented by user action constants. Typical user action constants are as follows:

Constant	*Value*	*Description*
`kUserActionCancel`	`1`	The user clicked the Cancel button.
`kUserActionOpen`	`2`	The user clicked the Open button in an Open dialog.
`kUserActionSaveAs`	`3`	The user clicked the Save button in a Save Location dialog.
`kUserActionChoose`	`4`	The user clicked the Choose button in a Choose dialog

As an alternative to calling `NavDialogGetUserAction`, you can extract the user action from the `userAction` field of the `NavCBRec` structure.

After determining the user action, your event handling function should take the appropriate action. For example, if the Open button was clicked, your event handling function should proceed to open the file, or files, selected by the user in the Open dialog.

You should always call the function `NavCompleteSave` to complete any save operation. Amongst other things, `NavCompleteSave` performs any needed translation.

kNavCBTerminate Message Received

When the `kNavCBTerminate` message is received, your event handler should call `NavDialogDispose` to dispose of the dialog reference.

Event Handling in Primary Alerts

Your event handling function for the primary alerts should be declared in the same way as that for the event handling function for the primary dialogs.

When the `kNavCBUserAction` message is received, your application should call `NavDialogGetUserAction` to determine the user action, and then take the appropriate action. The user action constants relevant to the primary alerts are as follows:

Constant	*Value*	*Description*
`kUserActionCancel`	`1`	The user clicked the Cancel button.
`kUserActionSaveChanges`	`6`	The user clicked the Save button in a Save Changes alert.
`kUserActionDontSaveChanges`	`7`	The user clicked the Don't Save button in a Save Changes alert.
`kUserActionDiscardChanges`	`8`	The user clicked the OK button in a Discard Changes alert.

When the appropriate action has been taken, your event handler should call `NavDialogDispose` to dispose of the dialog reference.

Other Application-Defined (Callback) Functions

Application-Defined Object Filtering

If your application needs simple, straightforward object filtering, and as previously described, you simply pass a pointer to a structure of type `NavTypeList` to the relevant Navigation Services function (`NavCreateGetFileDialog` or `NavCreateChooseFileDialog`). If you want more specific filtering, you can provide an application-defined filter (callback) function. Filter functions give you more control over what can and cannot be displayed. You can pass a universal procedure pointer to your filter function in calls to the functions `NavCreateGetFileDialog`, `NavCreateChooseFileDialog`, `NavCreateChooseFolderDialog`, `NavCreateChooseVolumeDialog`, and `NavCreateChooseObjectDialog`.

You can use both a `NavTypeList` structure and a filter function for the Open and Choose a File dialogs if you wish, but be aware that your filter function is directly affected by the `NavTypeList` structure. For example, if the `NavTypeList` structure contains only `'TEXT'` and `'PICT'` types, only `'TEXT'` and `'PICT'` files will be passed into your filter function.

Your filter function should return `true` if an object is to be displayed. The following is an example of a filter function that allows only text files to be displayed:

```
Boolean  myNavFilterCallback(AEDesc *theItem, void *info,void *callBackUD,
                             NavFilterModes filterMode)
{
  OSErr                theErr = noErr;
  Boolean              display = true;
  NavFileOrFolderInfo  *theInfo;

  theInfo = (NavFileOrFolderInfo *) info;
  if(theItem->descriptorType == typeFSS)
    if(!theInfo->isFolder)
      if(theInfo->fileAndFolder.fileInfo.finderInfo.fdType != 'TEXT')
        display = false;

  return display;
}
```

Application-Defined (Callback) Previews

To override how previews are drawn and handled, you can create a preview function and pass a universal procedure pointer to it in the `inpreviewProc` parameter of the Navigation Services functions `NavCreateGetFileDialog`, `NavCreateChooseFileDialog` and `NavCreateChooseObjectDialog`:

```
Boolean  myPreviewProc(NavCBRecPtr callBackParms,void *callBackUD);
```

`callBackParms` A pointer to a `NavCBRec` structure that contains event data needed for your function to draw the preview.

`callBackUD` A value set by your application.

Return: `true` if your preview function successfully draws the file preview. If your preview function returns `false`, Navigation Services will display the preview if the file contains a valid `'pnot'` resource.

Main File Manager Constants, Data Types and Functions

Constants

Read/Write Permission

```
fsCurPerm    = 0
fsRdPerm     = 1
fsWrPerm     = 2
fsRdWrPerm   = 3
fsRdWrShPerm = 4
```

File Mark Positioning Modes

```
fsAtMark     = 0
fsFromStart  = 1
fsFromLEOF   = 2
fsFromMark   = 3
rdVerify     = 64
```

Data Types

File System Specification Structure

```
struct FSSpec
{
  short          vRefNum;     // Volume reference number.
  long           parID;       // Directory ID of parent directory.
  Str63          name;        // Filename or directory name.
} ;
typedef struct FSSpec FSSpec;
typedef FSSpec *FSSpecPtr, **FSSpecHandle;
```

File System Reference

```
struct FSRef
{
  UInt8 hidden[80];
}
```

File Information Structure

```
struct FInfo
{
  OSType         fdType;      // File type.
  OSType         fdCreator;   // File's creator.
  unsigned short fdFlags;     // Finder flags (fHasBundle, fInvisible, etc).
  Point          fdLocation;  // Position of top-left corner of file's icon.
  short          fdFldr;      // Folder containing file.
} ;
typedef struct FInfo FInfo;
```

Functions

Reading, Writing and Closing Files

```
OSErr  FSClose(short refNum);
```

```
OSErr  FSRead(short refNum,long *count,void *buffPtr);
OSErr  FSWrite(short refNum,long *count,const void *buffPtr);
```

Manipulating the File Mark

```
OSErr  GetFPos(short refNum,long *filePos);
OSErr  SetFPos(short refNum,short posMode,long posOff);
```

Manipulating the End-Of-File

```
OSErr  GetEOF(short refNum,long *logEOF);
OSErr  SetEOF(short refNum,long logEOF);
```

Opening and Creating Files

```
OSErr  FSpOpenDF(const FSSpec *spec,SInt8 permission,short *refNum);
OSErr  FSpOpenRF(const FSSpec *spec,SInt8 permission,short *refNum);
OSErr  FSpCreate(const FSSpec *spec,OSType creator,OSType fileType,ScriptCode scriptTag);
OSErr  FSCreateFileUnicode(const FSRef *parentRef,UniCharCount nameLength,const UniChar
*name,
       FSCatalogInfoBitmap whichInfo,const FSCatalogInfo *catalogInfo,FSRef *newRef,
       FSSpec *newSpec);
```

Deleting Files and Directories

```
OSErr  FSpDelete(const FSSpec *spec);
OSErr  FSDeleteObject(const FSRef *ref);
```

Exchanging Data in Two Files

```
OSErr  FSpExchangeFiles(const FSSpec *source,const FSSpec *dest);
OSErr  FSExchangeObjects(const FSRef *ref,const FSRef *destRef);
```

Creating File System Specifications and File System References

```
OSErr  FSMakeFSSpec(short vRefNum,long dirID,ConstStr255Param fileName,FSSpec *spec);
OSErr  FSpMakeFSRef(const FSSpec *source,FSRef *newRef);
OSErr  FSMakeFSRefUnicode(const FSRef *parentRef,UniCharCount nameLength,const UniChar
*name,
       TextEncoding textEncodingHint,FSRef *newRef)
```

Obtaining Volume Information

```
OSErr  GetVInfo(short drvNum,StringPtr volName,short *vRefNum,long *freeBytes);
OSErr  GetVRefNum(short fileRefNum,short *vRefNum);
```

Getting and Setting Finder Information

```
OSErr  FSpGetFInfo(FSSpec *spec,FInfo *fndrInfo);
OSErr  FSpSetFInfo(const FSSpec *spec,const FInfo *fndrInfo);
```

Relevant Resource Manager Functions

Creating and Opening Resource Files

```
void   FSpCreateResFile(const FSSpec *spec,OSType creator,OSType fileType,
       ScriptCode scriptTag);
short  FSpOpenResFile(const FSSpec *spec,SignedByte permission);
```

Relevant Finder Interface Functions

Find a Specified Folder

```
OSErr  FindFolder(short vRefNum,OSType folderType,Boolean createFolder,
       short *foundVRefNum,long *foundDirID)
```

Main Navigation Services Constants, Data Types, and Functions

Constants

Dialog Option Flags

```
kNavDefaultNavDlogOptions  = 0x000000E4
kNavNoTypePopup            = 0x00000001
kNavDontAutoTranslate      = 0x00000002
kNavDontAddTranslateItems  = 0x00000004
kNavAllFilesInPopup        = 0x00000010
kNavAllowStationery        = 0x00000020
kNavAllowPreviews          = 0x00000040
kNavAllowMultipleFiles     = 0x00000080
kNavAllowInvisibleFiles    = 0x00000100
kNavDontResolveAliases     = 0x00000200
kNavSelectDefaultLocation  = 0x00000400
kNavSelectAllReadableItem  = 0x00000800
kNavSupportPackages        = 0x00001000
kNavAllowOpenPackages      = 0x00002000
kNavDontAddRecents         = 0x00004000
kNavDontUseCustomFrame     = 0x00008000
kNavDontConfirmReplacement = 0x00010000
```

Event Messages

```
kNavCBEvent           = 0
kNavCBCustomize       = 1
kNavCBStart           = 2
kNavCBTerminate       = 3
kNavCBAdjustRect      = 4
kNavCBNewLocation     = 5
kNavCBShowDesktop     = 6
kNavCBSelectEntry     = 7
kNavCBPopupMenuSelect = 8
kNavCBAccept          = 9
kNavCBCancel          = 10
kNavCBAdjustPreview   = 11
kNavCBUserAction      = 12
kNavCBOpenSelection   = (long) 0x80000000
```

User Action

```
kNavUserActionNone            = 0
kNavUserActionCancel          = 1
kNavUserActionOpen            = 2
kNavUserActionSaveAs          = 3
kNavUserActionChoose          = 4
kNavUserActionNewFolder       = 5
kNavUserActionSaveChanges     = 6
kNavUserActionDontSaveChanges = 7
kNavUserActionDiscardChanges  = 8
```

Save Changes Action

```
kNavSaveChangesClosingDocument     = 1
kNavSaveChangesQuittingApplication = 2
```

Data Types

```
typedef struct __NavDialog *NavDialogRef;
typedef UInt32              NavDialogOptionFlags;
typedef SInt32              NavEventCallbackMessage;
typedef void               *NavCallBackUserData;
typedef UInt32              NavUserAction;
typedef UInt32              NavAskSaveChangesAction;
```

NavDialogCreationOptions

```
struct NavDialogCreationOptions
{
  UInt16              version;
  NavDialogOptionFlags optionFlags;
  Point               location;
  CFStringRef         clientName;
  CFStringRef         windowTitle;
  CFStringRef         actionButtonLabel;
  CFStringRef         cancelButtonLabel;
  CFStringRef         saveFileName;
  CFStringRef         message;
  UInt32              preferenceKey;
  CFArrayRef          popupExtension;
  WindowModality      modality;
  WindowRef           parentWindow;
  char                reserved[16];
} ;
typedef struct NavDialogCreationOptions NavDialogCreationOptions;
```

NavTypeList

```
struct NavTypeList
{
  OSType componentSignature;
  short  reserved;
  short  osTypeCount;
  OSType osType[1];
} ;
typedef struct NavTypeList NavTypeList;
typedef NavTypeList *NavTypeListPtr;
typedef NavTypeListPtr *NavTypeListHandle;
```

NavCBRec

```
struct NavCBRec
{
  UInt16        version;
  NavDialogRef  context;
  WindowRef     window;
  Rect          customRect;
  Rect          previewRect;
  NavEventData  eventData;
  NavUserAction userAction;
  char          reserved[218];
} ;
typedef struct NavCBRec NavCBRec;
typedef NavCBRec *NavCBRecPtr;
```

NavReplyRecord

```
struct NavReplyRecord
{
  UInt16                      version;
  Boolean                     validRecord;
  Boolean                     replacing;
```

```
  Boolean                       isStationery;
  Boolean                       translationNeeded;
  AEDescList                    selection;
  ScriptCode                    keyScript;
  FileTranslationSpecArrayHandle fileTranslation;
  UInt32                        reserved1;
  CFStringRef                   saveFileName;
  char                          reserved[227];
} ;
typedef struct NavReplyRecord  NavReplyRecord;
```

Functions

Initializing the NavDialogCreationOptions Structure

```
OSStatus      NavGetDefaultDialogCreationOptions(NavDialogCreationOptions *outOptions);
```

Creating and Disposing Of Navigation Services Dialogs

```
OSStatus      NavCreateGetFileDialog(const NavDialogCreationOptions *inOptions,
              NavTypeListHandle inTypeList,NavEventUPP inEventProc,NavPreviewUPP
inPreviewProc,
              NavObjectFilterUPP inFilterProc,void *inClientData,NavDialogRef *outDialog);
OSStatus      NavCreatePutFileDialog(const NavDialogCreationOptions *inOptions,
              OSType inFileType,OSType inFileCreator,NavEventUPP inEventProc,
              void *inClientData,NavDialogRef *outDialog);
OSStatus      NavCreateAskSaveChangesDialog(const NavDialogCreationOptions *inOptions,
              NavAskSaveChangesAction inAction,NavEventUPP inEventProc,void *inClientData,
              NavDialogRef *outDialog);
OSStatus      NavCreateAskDiscardChangesDialog(const NavDialogCreationOptions *inOptions,
              NavEventUPP inEventProc,void *inClientData,NavDialogRef *outDialog);
OSStatus      NavCreateChooseFileDialog(const NavDialogCreationOptions *inOptions,
              NavTypeListHandle inTypeList,NavEventUPP inEventProc,NavPreviewUPP
inPreviewProc,
              NavObjectFilterUPP inFilterProc,void *inClientData,NavDialogRef *outDialog);
OSStatus      NavCreateChooseVolumeDialog(const NavDialogCreationOptions *inOptions,
              NavEventUPP inEventProc,NavObjectFilterUPP inFilterProc,void *inClientData,
              NavDialogRef *outDialog);
OSStatus      NavCreateChooseObjectDialog(const NavDialogCreationOptions *inOptions,
              NavEventUPP inEventProc,NavPreviewUPP inPreviewProc,
              NavObjectFilterUPP inFilterProc,void *inClientData,NavDialogRef *outDialog);
void          NavDialogDispose(NavDialogRef inDialog);
```

Displaying and Running a Navigation Services Dialog

```
OSStatus      NavDialogRun(NavDialogRef inDialog);
```

Filling In and Disposing Of NavReplyRecord Structures

```
OSStatus      NavDialogGetReply(NavDialogRef inDialog,NavReplyRecord *outReply);
OSErr         NavDisposeReply(NavReplyRecord *reply);
```

Getting the User Action

```
NavUserAction NavDialogGetUserAction(NavDialogRef inDialog);
```

Getting and Setting the Save File Name

```
CFStringRef   NavDialogGetSaveFileName(NavDialogRef inPutFileDialog);
OSStatus      NavDialogSetSaveFileName(NavDialogRef inPutFileDialog,CFStringRef inFileName);
```

Completing a Save Operation

```
OSErr         NavCompleteSave(NavReplyRecord *reply,NavTranslationOptions howToTranslate);
```

Getting the Window In Which a Navigation Services Dialog Appears

```
WindowRef     NavDialogGetWindow(NavDialogRef inDialog);
```

Creating New Folders

```
OSStatus      NavCreateNewFolderDialog(const NavDialogCreationOptions *inOptions,
              NavEventUPP inEventProc,void *inClientData,NavDialogRef *outDialog);
```

Creating Previews

```
OSErr         NavCreatePreview(AEDesc *theObject,OSType previewDataType,
              const void *previewData,Size  previewDataSize);
```

Creating and Disposing of Universal Procedure Pointers

```
NavEventUPP        NewNavEventUPP(NavEventProcPtr userRoutine);
NavPreviewUPP      NewNavPreviewUPP(NavPreviewProcPtr userRoutine);
NavObjectFilterUPP NewNavObjectFilterUPP(NavObjectFilterProcPtr userRoutine);
void               DisposeNavEventUPP(NavEventUPP userUPP);
void               DisposeNavPreviewUPP(NavPreviewUPP userUPP);
void               DisposeNavObjectFilterUPP(NavObjectFilterUPP userUPP);
```

Application-Defined (Callback) Functions-Event Handling, Previews, and Filters

```
void               myNavEventFunction(NavEventCallbackMessage callBackSelector,
                   NavCBRecPtr callBackParms,void *callBackUD);
Boolean            myNavPreviewFunction(NavCBRecPtr callBackParms,void *callBackUD);
Boolean            myNavObjectFilterFunction(AEDesc *theItem,void *info,void *callBackUD,
                   NavFilterModes filterMode);
```

Demonstration Program Files Listing

```
//
**********************************************************************************************
// Files.h                                                                    CARBON EVENT
MODEL
//
**********************************************************************************************
//
// This program demonstrates:
//
// •  File operations associated with:
//
//    •  The user invoking the Open..., Close, Save, Save As..., Revert, and Quit commands
of a
//      typical application.
//
//    •  Handling of the required Apple events Open Application, Re-open Application, Open
//      Documents, Print Documents, and Quit Application.
//
// •  File synchronisation.
//
// •  The creation, display, and handling of Open, Save Location, Choose a Folder, Save
//    Changes, Discard Changes, and Review Unsaved dialogs and alerts using the new model
//    introduced with Navigation Services 3.0.
//
// To keep the code not specifically related to files and file-handling to a minimum, an
item
// is included in the Demonstration menu which allows the user to simulate "touching" a
window
// (that is, modifying the contents of the associated document). Choosing the first menu
item
// in this menu sets the window-touched flag in the window's document structure to true and
// draws the text "WINDOW TOUCHED" in the window in a large font size, this latter so that
the
// user can keep track of which windows have been "touched".
//
// This program is also, in part, an extension of the demonstration program Windows2 in that
// it also demonstrates certain file-related Window Manager features introduced with the Mac
// OS 8.5 Window Manager. These features are:
//
// •  Window proxy icons.
//
// •  Window path pop-up menus.
//
// Those sections of the source code relating to these features are identified with ///// at
// the right of each line.
//
// The program utilises the following resources:
//
// •  A 'plst' resource containing an information property list which provides information
//    to the Mac OS X Finder.
//
// •  An 'MBAR' resource, and 'MENU' and 'xmnu' resources for Apple, File, Edit and
//    Demonstration menus (preload, non-purgeable).
//
// •  A 'STR ' resource containing the "missing application name" string, which is copied to
//    all document files created by the program.
//
// •  'STR#' resources (purgeable) containing error strings, the application's name (for
//    certain Navigation Services functions), and a message  string for the Choose a Folder
//    dialog.
```

```
//
// •  An 'open' resource (purgeable) containing the file type list for the Open dialog.
//
// •  A 'kind' resource (purgeable) describing file types, which is used by Navigation
//    Services to  build the native file types section of the Show pop-up menu in the Open
//    dialog.
//
// •  Two 'pnot' resources (purgeable) which, together with an associated 'PICT' resource
//    (purgeable) and a 'TEXT' resource created by the program, provide the previews for
//    the PICT and, on Mac OS 8/9, TEXT files.
//
// •  The 'BNDL' resource (non-purgeable), 'FREF' resources (non-purgeable), signature
//    resource (non-purgeable), and icon family resources (purgeable), required to support
the
//    built application on Mac OS 8/9.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
//
*****************************************************************************************

// .....................................................................
includes

#include <Carbon.h>

// .....................................................................
defines

#define rMenubar                  128
#define mAppleApplication         128
#define  Apple_About              'abou'
#define mFile                     129
#define  File_New                 'new '
#define  File_Open                'open'
#define  File_Close               'clos'
#define  File_Save                'save'
#define  File_SaveAs              'sava'
#define  File_Revert              'reve'
#define  File_Quit                'quit'
#define  iQuit                    12
#define mDemonstration            131
#define  Demo_TouchWindow         'touc'
#define  Demo_ChooseAFolderDialog 'choo'
#define rErrorStrings             128
#define  eInstallHandler          1000
#define  eMaxWindows              1001
#define  eCantFindFinderProcess
1002                                                  /////
#define rMiscStrings              129
#define  sApplicationName         1
#define  sChooseAFolder           2
#define rOpenResource             128
#define kMaxWindows               10
#define kFileCreator              'Kjbb'
#define kFileTypeTEXT             'TEXT'
#define kFileTypePICT             'PICT'
#define kOpen                     0
#define kPrint                    1
#define MIN(a,b)                  ((a) < (b) ? (a) : (b))

// .....................................................................
typedefs

typedef struct
```

```
{
  TEHandle     editStrucHdl;
  PicHandle    pictureHdl;
  SInt16       fileRefNum;
  FSSpec       fileFSSpec;
  AliasHandle  aliasHdl;
  Boolean      windowTouched;
  NavDialogRef modalToWindowNavDialogRef;
  NavEventUPP  askSaveDiscardEventFunctionUPP;
  Boolean      isAskSaveChangesDialog;
}   docStructure, *docStructurePointer, **docStructureHandle;

// ..................................................................... function
prototypes

void      main                          (void);
void      eventLoop                     (void);
void      doPreliminaries               (void);
void      doInstallAEHandlers           (void);
OSStatus  appEventHandler               (EventHandlerCallRef,EventRef,void *);
OSStatus  windowEventHandler            (EventHandlerCallRef,EventRef,void *);
void      doIdle                        (void);
void      doDrawContent                 (WindowRef);
void      doMenuChoice                  (MenuCommand);
void      doAdjustMenus                 (void);
void      doErrorAlert                  (SInt16);
void      doCopyPString                 (Str255,Str255);
void      doConcatPStrings              (Str255,Str255);
void      doTouchWindow                 (void);
OSErr     openAppEventHandler           (AppleEvent *,AppleEvent *,SInt32);
OSErr     reopenAppEventHandler         (AppleEvent *,AppleEvent *,SInt32);
OSErr     openAndPrintDocsEventHandler  (AppleEvent *,AppleEvent *,SInt32);
OSErr     quitAppEventHandler           (AppleEvent *,AppleEvent *,SInt32);
OSErr     doHasGotRequiredParams        (AppleEvent *);
SInt16    doReviewChangesAlert          (SInt16);

OSErr     doNewCommand                  (void);
OSErr     doOpenCommand                 (void);
OSErr     doCloseCommand                (NavAskSaveChangesAction);
OSErr     doSaveCommand                 (void);
OSErr     doSaveAsCommand               (void);
OSErr     doRevertCommand               (void);

OSErr     doNewDocWindow                (Boolean,OSType,WindowRef *);
EventHandlerUPP  doGetHandlerUPP        (void);
OSErr     doCloseDocWindow              (WindowRef);
OSErr     doOpenFile                    (FSSpec,OSType);
OSErr     doReadTextFile                (WindowRef);
OSErr     doReadPictFile                (WindowRef);
OSErr     doCreateAskSaveChangesDialog  (WindowRef,docStructureHandle,
NavAskSaveChangesAction);
OSErr     doSaveUsingFSSpec             (WindowRef,NavReplyRecord *);
OSErr     doSaveUsingFSRef              (WindowRef,NavReplyRecord *);
OSErr     doWriteFile                   (WindowRef);
OSErr     doWriteTextData               (WindowRef,SInt16);
OSErr     doWritePictData               (WindowRef,SInt16);

void  getFilePutFileEventFunction   (NavEventCallbackMessage,NavCBRecPtr,
NavCallBackUserData);
void  askSaveDiscardEventFunction   (NavEventCallbackMessage,NavCBRecPtr,
NavCallBackUserData);

OSErr     doCopyResources               (WindowRef);
OSErr     doCopyAResource               (ResType,SInt16,SInt16,SInt16);

void      doSynchroniseFiles            (void);
```

```
OSErr     doChooseAFolderDialog        (void);

//
**********************************************************************************************
// Files.c
//
**********************************************************************************************

// ..........................................................................
includes

#include "Files.h"

// ..................................................................... global
variables

Boolean     gRunningOnX = false;
SInt16      gAppResFileRefNum;
NavEventUPP gGetFilePutFileEventFunctionUPP ;
Boolean     gQuittingApplication = false;

extern SInt16 gCurrentNumberOfWindows;
extern Rect   gDestRect,gViewRect;

// ************************************************************************************
main

void  main(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  EventTypeSpec applicationEvents[] = {  {  kEventClassApplication,
kEventAppActivated    } ,
                                       {  kEventClassCommand,     kEventProcessCommand  } ,
                                       {  kEventClassMenu,
kEventMenuEnableItems }  } ;
  EventLoopTimerRef timerRef;

  // ..................................................................... do
preliminaries

  doPreliminaries();

  // ................................. save application's resource file file reference
number

  gAppResFileRefNum = CurResFile();
  // ..................................................................... set up menu bar and
menus

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    doErrorAlert(MemError());
  SetMenuBar(menubarHdl);
  DrawMenuBar();

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
    }
```

```
    gRunningOnX = true;
  }
  else
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
      SetMenuItemCommandID(menuRef,iQuit,kHICommandQuit);
  }

  // ................................................ install required Apple event
handlers

  doInstallAEHandlers();

  // ..................................................... install application event
handler

  InstallApplicationEventHandler(NewEventHandlerUPP((EventHandlerProcPtr) appEventHandler),
                                 GetEventTypeCount(applicationEvents),applicationEvents,
                                 0,NULL);

  // ............................................ install a timer (for file
synchronisation)

  InstallEventLoopTimer(GetCurrentEventLoop(),0,TicksToEventTime(15),
                        NewEventLoopTimerUPP((EventLoopTimerProcPtr) doIdle),NULL,
                        &timerRef);

  // .... get universal procedure pointer to main Navigation Services services event
function

  gGetFilePutFileEventFunctionUPP  =
                                NewNavEventUPP((NavEventProcPtr)
getFilePutFileEventFunction);

  // ........................................................... run application event
loop

  RunApplicationEventLoop();
}
// ********************************************************************************
doPreliminaries

void  doPreliminaries(void)
{
  MoreMasterPointers(448);
  InitCursor();
}

// ********************************************************************************
doInstallAEHandlers

void  doInstallAEHandlers(void)
{
  OSErr osError;

  osError = AEInstallEventHandler(kCoreEventClass,kAEOpenApplication,
                  NewAEEventHandlerUPP((AEEventHandlerProcPtr) openAppEventHandler),
                  0L,false);
  if(osError != noErr)  doErrorAlert(eInstallHandler);

  osError = AEInstallEventHandler(kCoreEventClass,kAEReopenApplication,
                  NewAEEventHandlerUPP((AEEventHandlerProcPtr) reopenAppEventHandler),
                  0L,false);
  if(osError != noErr)  doErrorAlert(eInstallHandler);
```

```
  osError = AEInstallEventHandler(kCoreEventClass,kAEOpenDocuments,
                  NewAEEventHandlerUPP((AEEventHandlerProcPtr)
openAndPrintDocsEventHandler),
                  kOpen,false);
  if(osError != noErr)  doErrorAlert(eInstallHandler);

  osError = AEInstallEventHandler(kCoreEventClass,kAEPrintDocuments,
                  NewAEEventHandlerUPP((AEEventHandlerProcPtr)
openAndPrintDocsEventHandler),
                  kPrint,false);
  if(osError != noErr)  doErrorAlert(eInstallHandler);

  osError = AEInstallEventHandler(kCoreEventClass,kAEQuitApplication,
                  NewAEEventHandlerUPP((AEEventHandlerProcPtr) quitAppEventHandler),
                  0L,false);
  if(osError != noErr)  doErrorAlert(eInstallHandler);
}

// ****************************************************************************
appEventHandler

OSStatus  appEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                          void * userData)
{
  OSStatus      result = eventNotHandledErr;
  UInt32        eventClass;
  UInt32        eventKind;
  HICommand     hiCommand;
  MenuID        menuID;
  MenuItemIndex menuItem;

  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);
  switch(eventClass)
  {
    case kEventClassApplication:
      if(eventKind == kEventAppActivated)
        SetThemeCursor(kThemeArrowCursor);
      break;

    case kEventClassCommand:
      if(eventKind == kEventProcessCommand)
      {
        GetEventParameter(eventRef,kEventParamDirectObject,typeHICommand,NULL,
                          sizeof(HICommand),NULL,&hiCommand);
        menuID = GetMenuID(hiCommand.menu.menuRef);
        menuItem = hiCommand.menu.menuItemIndex;
        if((hiCommand.commandID != kHICommandQuit) &&
           (menuID >= mAppleApplication && menuID <= mDemonstration))
        {
          doMenuChoice(hiCommand.commandID);
          result = noErr;
        }
      }
      break;

    case kEventClassMenu:
      if(eventKind == kEventMenuEnableItems)
      {
        doAdjustMenus();
        result = noErr;
      }
      break;
  }
```

```
  return result;
}

// ************************************************************************
windowEventHandler

OSStatus  windowEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                             void* userData)
{
  OSStatus  result = eventNotHandledErr;
  UInt32    eventClass;
  UInt32    eventKind;
  WindowRef windowRef;

  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);

  switch(eventClass)
  {
    case kEventClassWindow:
      GetEventParameter(eventRef,kEventParamDirectObject,typeWindowRef,NULL,sizeof
(windowRef),
                        NULL,&windowRef);
      switch(eventKind)
      {
        case kEventWindowDrawContent:
          doDrawContent(windowRef);
          result = noErr;
          break;

        case kEventWindowClose:
          if(gQuittingApplication)
            doCloseCommand(kNavSaveChangesQuittingApplication);
          else
            doCloseCommand(kNavSaveChangesClosingDocument);
          result = noErr;
          break;
      }
      break;
  }

  return result;
}

// ************************************************************************************
doIdle

void  doIdle(void)
{
  if(GetWindowKind(FrontWindow()) == kApplicationWindowKind)
    doSynchroniseFiles();
}

// ************************************************************************************
doUpdate

void  doDrawContent(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  GrafPtr            oldPort;
  Rect               destRect;

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);
```

```
  if((*docStrucHdl)->pictureHdl)
  {
    destRect = (*(*docStrucHdl)->pictureHdl)->picFrame;
    OffsetRect(&destRect,170,54);
    HLock((Handle) (*docStrucHdl)->pictureHdl);
    DrawPicture((*docStrucHdl)->pictureHdl,&destRect);
    HUnlock((Handle) (*docStrucHdl)->pictureHdl);
  }
  else if((*docStrucHdl)->editStrucHdl)
  {
    HLock((Handle) (*docStrucHdl)->editStrucHdl);
    TEUpdate(&gDestRect,(*docStrucHdl)->editStrucHdl);
    HUnlock((Handle) (*docStrucHdl)->editStrucHdl);
  }
  if((*docStrucHdl)->windowTouched)
  {
    TextSize(48);
    MoveTo(30,170);
    DrawString("\pWINDOW TOUCHED");
    TextSize(12);
  }

  SetPort(oldPort);
}

// ****************************************************************************
doMenuChoice

void  doMenuChoice(MenuCommand commandID)
{
  OSErr osError = noErr;

  switch(commandID)
  {
    // ..................................................... Apple/Application
menu

    case Apple_About:
      SysBeep(10);
      break;

    // ..................................................................... File
menu

    case File_New:
      if(osError = doNewCommand())
        doErrorAlert(osError);
      break;

    case File_Open:
      if(osError = doOpenCommand() && osError == opWrErr)
        doErrorAlert(osError);
      break;

    case File_Close:
      if(osError = doCloseCommand(kNavSaveChangesClosingDocument))
        doErrorAlert(osError);
      break;

    case File_Save:
      if(osError = doSaveCommand())
        doErrorAlert(osError);
      break;

    case File_SaveAs:
```

```
      if(osError = doSaveAsCommand())
        doErrorAlert(osError);
      break;

    case File_Revert:
      if(osError = doRevertCommand())
        doErrorAlert(osError);
      break;
    // ..................................................................... Demonstration
menu

    case Demo_TouchWindow:
      doTouchWindow();
      break;

    case Demo_ChooseAFolderDialog:
      if(osError = doChooseAFolderDialog())
        doErrorAlert(osError);
      break;
  }
}

// ****************************************************************************
doAdjustMenus

void  doAdjustMenus(void)
{
  OSErr              osError;
  MenuRef            menuRef;
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;

  if(gCurrentNumberOfWindows > 0)
  {
    if(gRunningOnX)
    {
      if((osError = GetSheetWindowParent(FrontWindow(),&windowRef)) == noErr)
      {
        menuRef = GetMenuRef(mFile);
        DisableMenuCommand(menuRef,File_Close);
        DisableMenuCommand(menuRef,File_Save);
        DisableMenuCommand(menuRef,File_SaveAs);
        DisableMenuCommand(menuRef,File_Revert);
        menuRef = GetMenuRef(mDemonstration);
        DisableMenuCommand(menuRef,Demo_TouchWindow);
        return;
      }
      else
        windowRef = FrontWindow();
    }
    else
      windowRef = FrontWindow();

    if(GetWindowKind(windowRef) == kApplicationWindowKind)
    {
      docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

      menuRef = GetMenuRef(mFile);
      EnableMenuCommand(menuRef,File_Close);
      if((*docStrucHdl)->windowTouched)
      {
        EnableMenuCommand(menuRef,File_Save);
        EnableMenuCommand(menuRef,File_Revert);
      }
      else
      {
```

```
        DisableMenuCommand(menuRef,File_Save);
        DisableMenuCommand(menuRef,File_Revert);
      }

      if(((*docStrucHdl)->pictureHdl != NULL) ||
         ((*(*docStrucHdl)->editStrucHdl)->teLength > 0))
        EnableMenuCommand(menuRef,File_SaveAs);
      else
        DisableMenuCommand(menuRef,File_SaveAs);

      menuRef = GetMenuRef(mDemonstration);

      if(((*docStrucHdl)->pictureHdl != NULL) ||
         ((*(*docStrucHdl)->editStrucHdl)->teLength > 0))
      {
        if((*docStrucHdl)->windowTouched == false)
          EnableMenuCommand(menuRef,Demo_TouchWindow);
        else
          DisableMenuCommand(menuRef,Demo_TouchWindow);
      }
      else
        DisableMenuCommand(menuRef,Demo_TouchWindow);
    }
  }
  else
  {
    menuRef = GetMenuRef(mFile);
    DisableMenuCommand(menuRef,File_Close);
    DisableMenuCommand(menuRef,File_Save);
    DisableMenuCommand(menuRef,File_SaveAs);
    DisableMenuCommand(menuRef,File_Revert);
    menuRef = GetMenuRef(mDemonstration);
    DisableMenuCommand(menuRef,Demo_TouchWindow);
  }

  DrawMenuBar();
}

// ************************************************************************
doErrorAlert

void  doErrorAlert(SInt16 errorCode)
{
  Str255 errorString, theString;
  SInt16 itemHit;

  if(errorCode == eInstallHandler)
    GetIndString(errorString,rErrorStrings,1);
  else if(errorCode == eMaxWindows)
    GetIndString(errorString,rErrorStrings,2);
  else if(errorCode == eCantFindFinderProcess)
    GetIndString(errorString,rErrorStrings,3);
  else if(errorCode == opWrErr)
    GetIndString(errorString,rErrorStrings,4);
  else
  {
    GetIndString(errorString,rErrorStrings,5);
    NumToString((SInt32) errorCode,theString);
    doConcatPStrings(errorString,theString);
  }

  if(errorCode != memFullErr)
  {
    StandardAlert(kAlertCautionAlert,errorString,NULL,NULL,&itemHit);
  }
  else
```

```
  {
    StandardAlert(kAlertStopAlert,errorString,NULL,NULL,&itemHit);
    ExitToShell();
  }
}

// *****************************************************************************
doCopyPString

void  doCopyPString(Str255 sourceString,Str255 destinationString)
{
  SInt16 stringLength;

  stringLength = sourceString[0];
  BlockMove(sourceString + 1,destinationString + 1,stringLength);
  destinationString[0] = stringLength;
}

// *****************************************************************************
doConcatPStrings

void  doConcatPStrings(Str255 targetString,Str255 appendString)
{
  SInt16 appendLength;

  appendLength = MIN(appendString[0],255 - targetString[0]);

  if(appendLength > 0)
  {
    BlockMoveData(appendString+1,targetString+targetString[0]+1,(SInt32) appendLength);
    targetString[0] += appendLength;
  }
}

// *****************************************************************************
doTouchWindow

void  doTouchWindow(void)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  windowRef = FrontWindow();
  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

  SetPortWindowPort(windowRef);

  TextSize(48);
  MoveTo(30,170);
  DrawString("\pWINDOW TOUCHED");
  TextSize(12);

  (*docStrucHdl)->windowTouched = true;

  SetWindowModified(windowRef,
true);                                                  /////
}

// *****************************************************************************
openAppEventHandler

OSErr  openAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefCon)
{
  OSErr osError;

  osError = doHasGotRequiredParams(appEvent);
  if(osError == noErr)
```

```
    osError = doNewCommand();

  return osError;
}

// ********************************************************************
reopenAppEventHandler

OSErr  reopenAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,
                             SInt32 handlerRefCon)
{
  OSErr osError;

  osError = doHasGotRequiredParams(appEvent);
  if(osError == noErr)
    if(!FrontWindow())
      osError = doNewCommand();

  return osError;
}

// ************************************************************
openAndPrintDocsEventHandler

OSErr  openAndPrintDocsEventHandler(AppleEvent *appEvent,AppleEvent *reply,
                                    SInt32 handlerRefcon)
{
  FSSpec     fileSpec;
  AEDescList docList;
  OSErr      osError, ignoreErr;
  SInt32     index, numberOfItems;
  Size       actualSize;
  AEKeyword  keyWord;
  DescType   returnedType;
  FInfo      fileInfo;
  osError = AEGetParamDesc(appEvent,keyDirectObject,typeAEList,&docList);

  if(osError == noErr)
  {
    osError = doHasGotRequiredParams(appEvent);
    if(osError == noErr)
    {
      osError = AECountItems(&docList,&numberOfItems);
      if(osError == noErr)
      {
        for(index=1;index<=numberOfItems;index++)
        {
          osError = AEGetNthPtr(&docList,index,typeFSS,&keyWord,&returnedType,
                                &fileSpec,sizeof(fileSpec),&actualSize);
          if(osError == noErr)
          {
            osError = FSpGetFInfo(&fileSpec,&fileInfo);
            if(osError == noErr)
            {
              if(osError = doOpenFile(fileSpec,fileInfo.fdType))
                doErrorAlert(osError);

              if(osError == noErr && handlerRefcon == kPrint)
              {
                // Call printing function here
              }
            }
          }
          else
            doErrorAlert(osError);
        }
```

```
      }
    }
    else
      doErrorAlert(osError);

    ignoreErr = AEDisposeDesc(&docList);
  }
  else
    doErrorAlert(osError);

  return osError;
}

// *********************************************************************
quitAppEventHandler

OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  OSErr              osError;
  WindowRef          windowRef, previousWindowRef;
  docStructureHandle docStrucHdl;
  SInt16             touchedWindowsCount = 0;
  EventRef           eventRef;
  EventTargetRef     eventTargetRef;
  SInt16             itemHit;
  osError = doHasGotRequiredParams(appEvent);
  if(osError == noErr)
  {
    if(FrontWindow())
    {
      // .................. if any window has a sheet, bring to front, play system alert
sound, and return

      windowRef = GetFrontWindowOfClass(kSheetWindowClass,true);
      if(windowRef)
      {
        SelectWindow(windowRef);
        SysBeep(10);
        return noErr;
      }

      // ..................................................................... count touched
windows

      windowRef = FrontWindow();
      do
      {
        docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
        if((*docStrucHdl)->windowTouched == true)
          touchedWindowsCount++;
        previousWindowRef = windowRef;
      }  while(windowRef = GetNextWindowOfClass(previousWindowRef,kDocumentWindowClass,
true));

      // ............................................. if no touched windows, simply close
down

      if(touchedWindowsCount == 0)
        QuitApplicationEventLoop();

      // ............................... if touched windows are present, and if running on
OS X

      if(gRunningOnX)
      {
```

```
        // ...... if one touched window, cause Save Changes alert on that window, close all
others

        if(touchedWindowsCount == 1)
        {
          gQuittingApplication = true;
          CreateEvent(NULL,kEventClassWindow,kEventWindowClose,0,kEventAttributeNone,
                      &eventRef);
          eventTargetRef = GetWindowEventTarget(FrontWindow());
          SendEventToEventTarget(eventRef,eventTargetRef);
        }

      // ...... if more than one touched window, create Review Changes alert, handle button
clicks

      else if(touchedWindowsCount > 1)
      {
        itemHit = doReviewChangesAlert(touchedWindowsCount);
          if(itemHit == kAlertStdAlertOKButton)
          {
            gQuittingApplication = true;
            CreateEvent(NULL,kEventClassWindow,kEventWindowClose,0,kEventAttributeNone,
                        &eventRef);
            eventTargetRef = GetWindowEventTarget(FrontWindow());
            SendEventToEventTarget(eventRef,eventTargetRef);
          }
          else if(itemHit == kAlertStdAlertCancelButton)
            gQuittingApplication = false;
          else if(itemHit == kAlertStdAlertOtherButton)
            QuitApplicationEventLoop();
        }
      }

      // ........................... if touched windows are present, and if running on OS
8/9

      else
      {
        gQuittingApplication = true;
        CreateEvent(NULL,kEventClassWindow,kEventWindowClose,0,kEventAttributeNone,
                    &eventRef);
        eventTargetRef = GetWindowEventTarget(FrontWindow());
        SendEventToEventTarget(eventRef,eventTargetRef);
      }
    }
    else
      QuitApplicationEventLoop();
  }

  return osError;
}

// **********************************************************************
doHasGotRequiredParams

OSErr  doHasGotRequiredParams(AppleEvent *appEvent)
{
  DescType returnedType;
  Size     actualSize;
  OSErr    osError;

  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,
                              NULL,0,&actualSize);
  if(osError == errAEDescNotFound)
    osError = noErr;
  else if(osError == noErr)
```

```
    osError = errAEParamMissed;

  return osError;
}

// ********************************************************************
doReviewChangesAlert
SInt16  doReviewChangesAlert(SInt16 touchedWindowsCount)
{
  AlertStdCFStringAlertParamRec paramRec;
  Str255      messageText1 = "\pYou have ";
  Str255      messageText2 = "\p Files documents with unsaved changes. ";
  Str255      messageText3 = "\pDo you want to review these changes before quitting?";
  Str255      countString;
  CFStringRef messageText;
  CFStringRef informativeText =
              CFSTR("If you don't review your documents, all your changes will be lost.");
  DialogRef       dialogRef;
  DialogItemIndex itemHit;

  NumToString(touchedWindowsCount,countString);
  doConcatPStrings(messageText1,countString);
  doConcatPStrings(messageText1,messageText2);
  doConcatPStrings(messageText1,messageText3);
  messageText = CFStringCreateWithPascalString(NULL,messageText1,CFStringGetSystemEncoding
());

  GetStandardAlertDefaultParams(&paramRec,kStdCFStringAlertVersionOne);
  paramRec.movable      = true;
  paramRec.defaultText  = CFSTR("Review Changes...");
  paramRec.cancelText    = CFSTR("Cancel");
  paramRec.otherText    = CFSTR("Discard Changes");

  CreateStandardAlert(kAlertStopAlert,messageText,informativeText,&paramRec,&dialogRef);
  RunStandardAlert(dialogRef,NULL,&itemHit);

  if(messageText != NULL)
    CFRelease(messageText);

  return itemHit;
}

//
****************************************************************************************
// NewOpenCloseSave.c
//
****************************************************************************************

// ..........................................................................
includes

#include "Files.h"

// ..................................................................... global
variables

NavDialogRef gModalToApplicationNavDialogRef;
SInt16       gCurrentNumberOfWindows = 0;
Rect         gDestRect, gViewRect;
Boolean      gCloseDocWindow = false;

extern NavEventUPP gGetFilePutFileEventFunctionUPP;
extern SInt16      gAppResFileRefNum;
extern Boolean     gQuittingApplication;
extern Boolean     gRunningOnX;
// ********************************************************************
```

```
doNewCommand

OSErr  doNewCommand(void)
{
  WindowRef windowRef;
  OSErr     osError;
  OSType    documentType = kFileTypeTEXT;

  osError = doNewDocWindow(true,documentType,&windowRef);

  if(osError == noErr)
    SetWindowProxyCreatorAndType(windowRef,kFileCreator,documentType,
kUserDomain);       /////

  return osError;
}

// ************************************************************************
doOpenCommand

OSErr  doOpenCommand(void)
{
  OSErr                    osError = noErr;
  NavDialogCreationOptions dialogOptions;
  Str255                   applicationName;
  NavTypeListHandle        fileTypeListHdl = NULL;

  // .   ..... create application-modal Open
dialog

  osError = NavGetDefaultDialogCreationOptions(&dialogOptions);
  if(osError == noErr)
  {
    GetIndString(applicationName,rMiscStrings,sApplicationName);
    dialogOptions.clientName = CFStringCreateWithPascalString(NULL,applicationName,
                                                              CFStringGetSystemEncoding());
    dialogOptions.modality = kWindowModalityAppModal;
    fileTypeListHdl = (NavTypeListHandle) GetResource('open',rOpenResource);

    osError = NavCreateGetFileDialog(&dialogOptions,fileTypeListHdl,
                                     gGetFilePutFileEventFunctionUPP,NULL,NULL,NULL,
                                     &gModalToApplicationNavDialogRef);
    if(osError == noErr && gModalToApplicationNavDialogRef != NULL)
    {
      osError = NavDialogRun(gModalToApplicationNavDialogRef);
      if(osError != noErr)
      {
        NavDialogDispose(gModalToApplicationNavDialogRef);
        gModalToApplicationNavDialogRef = NULL;
      }
    }

    if(dialogOptions.clientName != NULL)
      CFRelease(dialogOptions.clientName);

    if(fileTypeListHdl != NULL)
      ReleaseResource((Handle) fileTypeListHdl);
  }
  return osError;
}

// ************************************************************************
doCloseCommand

OSErr  doCloseCommand(NavAskSaveChangesAction action)
{
```

```
  WindowRef          windowRef;
  SInt16             windowKind;
  docStructureHandle docStrucHdl;
  OSErr              osError = noErr;

  windowRef = FrontWindow();
  windowKind = GetWindowKind(windowRef);

  switch(windowKind)
  {
    case kApplicationWindowKind:
      docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

      // ........................... if window has unsaved changes, create Save Changes
alert

      if((*docStrucHdl)->windowTouched == true)
      {
        if(IsWindowCollapsed(windowRef))
          CollapseWindow(windowRef,false);

        osError = doCreateAskSaveChangesDialog(windowRef,docStrucHdl,action);
      }

      // ............................................... otherwise close file and clean
up

      else
        osError = doCloseDocWindow(windowRef);
      break;

    case kDialogWindowKind:
      // Hide or close modeless dialog, as required.
      break;
  }

  return osError;
}

// ****************************************************************************
doSaveCommand

OSErr  doSaveCommand(void)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  OSErr              osError = noErr;
  Rect               portRect;

  windowRef = FrontWindow();
  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  // ......... if the document has a file ref number, write the file, otherwise call
doSaveAsCommand

  if((*docStrucHdl)->fileRefNum)
  {
    osError = doWriteFile(windowRef);

    SetPortWindowPort(windowRef);
    GetWindowPortBounds(windowRef,&portRect);
    EraseRect(&portRect);
    InvalWindowRect(windowRef,&portRect);
  }
  else
    osError = doSaveAsCommand();
```

```
  if(osError ==
noErr)                                                                  /////
    SetWindowModified(windowRef,
false);                                                 /////

  return osError;
}

// **************************************************************************
doSaveAsCommand

OSErr  doSaveAsCommand(void)
{
  OSErr                    osError = noErr;
  NavDialogCreationOptions dialogOptions;
  WindowRef                windowRef;
  Str255                   windowTitle, applicationName;
  docStructureHandle       docStrucHdl;
  OSType                   fileType;

  // ........................................... create window-modal Save Location
dialog

  osError = NavGetDefaultDialogCreationOptions(&dialogOptions);
  if(osError == noErr)
  {
    dialogOptions.optionFlags |= kNavNoTypePopup;

    windowRef = FrontWindow();

    GetWTitle(windowRef,windowTitle);
    dialogOptions.saveFileName = CFStringCreateWithPascalString(NULL,windowTitle,
                                                                CFStringGetSystemEncoding
());
    GetIndString(applicationName,rMiscStrings,sApplicationName);
    dialogOptions.clientName = CFStringCreateWithPascalString(NULL,applicationName,
                                                            CFStringGetSystemEncoding());
    dialogOptions.parentWindow = windowRef;
    dialogOptions.modality = kWindowModalityWindowModal;

    docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
    if((*docStrucHdl)->editStrucHdl != NULL)
      fileType = kFileTypeTEXT;
    else if((*docStrucHdl)->pictureHdl != NULL)
      fileType = kFileTypePICT;
    HLock((Handle) docStrucHdl);

    osError = NavCreatePutFileDialog(&dialogOptions,fileType,kFileCreator,
                                     gGetFilePutFileEventFunctionUPP ,
                                     windowRef,&(*docStrucHdl)->modalToWindowNavDialogRef);
    HUnlock((Handle) docStrucHdl);

    if(osError == noErr && (*docStrucHdl)->modalToWindowNavDialogRef != NULL)
    {
      osError = NavDialogRun((*docStrucHdl)->modalToWindowNavDialogRef);
      if(osError != noErr)
      {
        NavDialogDispose((*docStrucHdl)->modalToWindowNavDialogRef);
        (*docStrucHdl)->modalToWindowNavDialogRef = NULL;
      }
    }

    if(dialogOptions.saveFileName != NULL)
      CFRelease(dialogOptions.saveFileName);
    if(dialogOptions.clientName != NULL)
      CFRelease(dialogOptions.clientName);
```

```
  }

  return osError;
}

// ************************************************************************
doRevertCommand

OSErr  doRevertCommand(void)
{
  OSErr                    osError = noErr;
  NavDialogCreationOptions dialogOptions;
  WindowRef                windowRef;
  Str255                   windowTitle;
  docStructureHandle       docStrucHdl;

  // ............................................ create window-modal Discard Changes
alert

  osError = NavGetDefaultDialogCreationOptions(&dialogOptions);
  if(osError == noErr)
  {
    windowRef = FrontWindow();

    GetWTitle(windowRef,windowTitle);
    dialogOptions.saveFileName = CFStringCreateWithPascalString(NULL,windowTitle,
                                                                CFStringGetSystemEncoding
());
    dialogOptions.parentWindow = windowRef;
    dialogOptions.modality = kWindowModalityWindowModal;

    docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
    if((*docStrucHdl)->askSaveDiscardEventFunctionUPP != NULL)
    {
      DisposeNavEventUPP((*docStrucHdl)->askSaveDiscardEventFunctionUPP);
      (*docStrucHdl)->askSaveDiscardEventFunctionUPP = NULL;
    }
    (*docStrucHdl)->askSaveDiscardEventFunctionUPP =
                          NewNavEventUPP((NavEventProcPtr) askSaveDiscardEventFunction);

    HLock((Handle) docStrucHdl);

    osError = NavCreateAskDiscardChangesDialog(&dialogOptions,
                                               (*docStrucHdl)-
>askSaveDiscardEventFunctionUPP,
                                               windowRef,
                                               &(*docStrucHdl)->modalToWindowNavDialogRef);
    HUnlock((Handle) docStrucHdl);

    if(osError == noErr && (*docStrucHdl)->modalToWindowNavDialogRef != NULL)
    {
      osError = NavDialogRun((*docStrucHdl)->modalToWindowNavDialogRef);
      if(osError != noErr)
      {
        NavDialogDispose((*docStrucHdl)->modalToWindowNavDialogRef);
        (*docStrucHdl)->modalToWindowNavDialogRef = NULL;
      }
    }

    if(dialogOptions.saveFileName != NULL)
      CFRelease(dialogOptions.saveFileName);
  }

  return osError;
}
```

```
// ****************************************************************************
doNewDocWindow

OSErr  doNewDocWindow(Boolean showWindow,OSType documentType,WindowRef * windowRef)
{
  OSStatus           osError;
  WindowAttributes   attributes = kWindowStandardHandlerAttribute |
                                  kWindowStandardDocumentAttributes;
  Rect               portRect, contentRect = {  0,0,300,500 } ;
  docStructureHandle docStrucHdl;
  EventTypeSpec      windowEvents[] = { {  kEventClassWindow, kEventWindowDrawContent  } ,
                                        {  kEventClassWindow, kEventWindowClose       } ,
                                        {  kEventClassWindow, kEventWindowClickDragRgn } ,
                                        {  kEventClassWindow,
kEventWindowPathSelect   }  } ;

  if(gCurrentNumberOfWindows == kMaxWindows)
    return eMaxWindows;

  // ..................... create window, change attributes, reposition, install event
handler

  osError = CreateNewWindow(kDocumentWindowClass,attributes,&contentRect,windowRef);
  if(osError != noErr)
    return osError;

  SetWTitle(*windowRef,"\puntitled");
  ChangeWindowAttributes(*windowRef,0,kWindowFullZoomAttribute | kWindowResizableAttribute);
  RepositionWindow(*windowRef,NULL,kWindowCascadeOnMainScreen);
  SetPortWindowPort(*windowRef);
  InstallWindowEventHandler(*windowRef,doGetHandlerUPP(),GetEventTypeCount(windowEvents),
                            windowEvents,0,NULL);

  // .... ............................................. attach document structure to
window

  if(!(docStrucHdl = (docStructureHandle) NewHandle(sizeof(docStructure))))
  {
    DisposeWindow(*windowRef);
    return MemError();
  }

  SetWRefCon(*windowRef,(SInt32) docStrucHdl);

  (*docStrucHdl)->editStrucHdl                  = NULL;
  (*docStrucHdl)->pictureHdl                    = NULL;
  (*docStrucHdl)->fileRefNum                    = 0;
  (*docStrucHdl)->aliasHdl                      =
NULL;                                  /////
  (*docStrucHdl)->windowTouched                 = false;
  (*docStrucHdl)->modalToWindowNavDialogRef     = NULL;
  (*docStrucHdl)->askSaveDiscardEventFunctionUPP = NULL;
  (*docStrucHdl)->isAskSaveChangesDialog        = false;

  // ........................................... if text document, create TextEdit
structure

  if(documentType == kFileTypeTEXT)
  {
    UseThemeFont(kThemeSmallSystemFont,smSystemScript);

    GetWindowPortBounds(*windowRef,&portRect);
    gDestRect = portRect;
    InsetRect(&gDestRect,6,6);
    gViewRect = gDestRect;
```

```
    MoveHHi((Handle) docStrucHdl);
    HLock((Handle) docStrucHdl);

    if(!((*docStrucHdl)->editStrucHdl = TENew(&gDestRect,&gViewRect)))
    {
      DisposeWindow(*windowRef);
      DisposeHandle((Handle) docStrucHdl);
      return MemError();
    }

    HUnlock((Handle) docStrucHdl);
  }

  // ........................................... show window and increment open windows
count

  if(showWindow)
    ShowWindow(*windowRef);

  gCurrentNumberOfWindows ++;

  return noErr;
}
// ************************************************************************
doGetHandlerUPP

EventHandlerUPP  doGetHandlerUPP(void)
{
  static EventHandlerUPP windowEventHandlerUPP;

  if(windowEventHandlerUPP == NULL)
    windowEventHandlerUPP = NewEventHandlerUPP((EventHandlerProcPtr) windowEventHandler);

  return windowEventHandlerUPP;
}

// ************************************************************************
doCloseDocWindow

OSErr  doCloseDocWindow(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  OSErr              osError = noErr;
  EventRef           eventRef;
  EventTargetRef     eventTargetRef;

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

  // ..................... close file, flush volume, dispose of window and associated
memory

  if((*docStrucHdl)->fileRefNum != 0)
  {
    if(!(osError = FSClose((*docStrucHdl)->fileRefNum)))
    {
      osError = FlushVol(NULL,(*docStrucHdl)->fileFSSpec.vRefNum);
      (*docStrucHdl)->fileRefNum = 0;
    }
  }

  if((*docStrucHdl)->editStrucHdl != NULL)
    TEDispose((*docStrucHdl)->editStrucHdl);
  if((*docStrucHdl)->pictureHdl != NULL)
    KillPicture((*docStrucHdl)->pictureHdl);

  DisposeHandle((Handle) docStrucHdl);
```

```
  DisposeWindow(windowRef);

  gCurrentNumberOfWindows --;

  // ..................................................................... if quitting
application

  if(gQuittingApplication)
  {
    if(FrontWindow() == NULL)
      QuitApplicationEventLoop();
    else
    {
      CreateEvent(NULL,kEventClassWindow,kEventWindowClose,0,kEventAttributeNone,
                  &eventRef);
      eventTargetRef = GetWindowEventTarget(FrontWindow());
      SendEventToEventTarget(eventRef,eventTargetRef);
    }
  }

  return osError;
}

// ****************************************************************************
doOpenFile

OSErr  doOpenFile(FSSpec fileSpec,OSType documentType)
{
  WindowRef          windowRef;
  OSErr              osError = noErr;
  SInt16             fileRefNum;
  docStructureHandle docStrucHdl;

  // ..................................................................... create new
window

  if(osError = doNewDocWindow(false,documentType,&windowRef))
    return osError;

  SetWTitle(windowRef,fileSpec.name);

  // ..................................................................... open file's data
fork

  if(osError = FSpOpenDF(&fileSpec,fsRdWrPerm,&fileRefNum))
  {
    DisposeWindow(windowRef);
    gCurrentNumberOfWindows --;
    return osError;
  }

  // ................... store file reference number and FSSpec in window's document
structure

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  (*docStrucHdl)->fileRefNum = fileRefNum;
  (*docStrucHdl)->fileFSSpec = fileSpec;

  // ..................................................................... read in the
file

  if(documentType == kFileTypeTEXT)
  {
    if(osError = doReadTextFile(windowRef))
      return osError;
  }
```

```
  else if(documentType == kFileTypePICT)
  {
    if(osError = doReadPictFile(windowRef))
      return osError;
  }
  // ..................................... set up window's proxy icon, and show
window

  SetWindowProxyFSSpec(windowRef,
&fileSpec);                                          /////
  GetWindowProxyAlias(windowRef,&((*docStrucHdl)-
>aliasHdl));                              /////
  SetWindowModified(windowRef,
false);                                                 /////

  ShowWindow(windowRef);

  return noErr;
}

// ************************************************************
doCreateAskSaveChangesDialog

OSErr  doCreateAskSaveChangesDialog(WindowRef windowRef,docStructureHandle docStrucHdl,
                                    NavAskSaveChangesAction action)
{
  OSErr                   osError = noErr;
  NavDialogCreationOptions dialogOptions;
  Str255                  windowTitle, applicationName;

  // ..................................... create window-modal Save Changes Changes
dialog

  osError = NavGetDefaultDialogCreationOptions(&dialogOptions);
  if(osError == noErr)
  {
    GetWTitle(windowRef,windowTitle);
    dialogOptions.saveFileName = CFStringCreateWithPascalString(NULL,windowTitle,
                                                                CFStringGetSystemEncoding
());

    GetIndString(applicationName,rMiscStrings,sApplicationName);
    dialogOptions.clientName = CFStringCreateWithPascalString(NULL,applicationName,
                                                              CFStringGetSystemEncoding());
    dialogOptions.parentWindow = windowRef;
    dialogOptions.modality = kWindowModalityWindowModal;

    if((*docStrucHdl)->askSaveDiscardEventFunctionUPP != NULL)
    {
      DisposeNavEventUPP((*docStrucHdl)->askSaveDiscardEventFunctionUPP);
      (*docStrucHdl)->askSaveDiscardEventFunctionUPP = NULL;
    }
    (*docStrucHdl)->askSaveDiscardEventFunctionUPP =
                                NewNavEventUPP((NavEventProcPtr)
askSaveDiscardEventFunction);

    HLock((Handle) docStrucHdl);

    osError = NavCreateAskSaveChangesDialog(&dialogOptions,action,
                                            (*docStrucHdl)->askSaveDiscardEventFunctionUPP,
                                            windowRef,
                                            &(*docStrucHdl)->modalToWindowNavDialogRef);
    HUnlock((Handle) docStrucHdl);

    if(osError == noErr && (*docStrucHdl)->modalToWindowNavDialogRef != NULL)
    {
```

```
      (*docStrucHdl)->isAskSaveChangesDialog = true;
      osError = NavDialogRun((*docStrucHdl)->modalToWindowNavDialogRef);
      if(osError != noErr)
      {
        NavDialogDispose((*docStrucHdl)->modalToWindowNavDialogRef);
        (*docStrucHdl)->modalToWindowNavDialogRef = NULL;
        (*docStrucHdl)->isAskSaveChangesDialog = false;
      }

      if(!gRunningOnX)
      {
        if(gCloseDocWindow)
        {
          osError = doCloseDocWindow(windowRef);
          if(osError != noErr)
            doErrorAlert(osError);
          gCloseDocWindow = false;
        }
      }
    }

    if(dialogOptions.saveFileName != NULL)
      CFRelease(dialogOptions.saveFileName);
    if(dialogOptions.clientName != NULL)
      CFRelease(dialogOptions.clientName);
  }

  return osError;
}

// ************************************************************************
doSaveUsingFSSpec

OSErr  doSaveUsingFSSpec(WindowRef windowRef,NavReplyRecord *navReplyStruc)
{
  OSErr              osError = noErr;
  AEKeyword          theKeyword;
  DescType           actualType;
  FSSpec             fileSpec;
  Size               actualSize;
  docStructureHandle docStrucHdl;
  OSType             fileType;
  CFStringRef        fileName;
  SInt16             fileRefNum;
  Rect               portRect;

  if((*navReplyStruc).validRecord)
  {

    // ..................................................................... get
FSSpec

    if((osError = AEGetNthPtr(&(*navReplyStruc).selection,1,typeFSS,&theKeyword,
                              &actualType,&fileSpec,sizeof(fileSpec),&actualSize)) ==
noErr)
    {
      docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
      // ............. get file name, convert to Pascal string, assign to name field of
FSSpec

      fileName = NavDialogGetSaveFileName((*docStrucHdl)->modalToWindowNavDialogRef);
      if(fileName != NULL)
        osError = CFStringGetPascalString(fileName,&fileSpec.name[0],sizeof(FSSpec),
                                          CFStringGetSystemEncoding());

      // ........................................ if not replacing, first create a new
```

```
file

      if(!((*navReplyStruc).replacing))
      {
        if((*docStrucHdl)->editStrucHdl != NULL)
          fileType = kFileTypeTEXT;
        else if((*docStrucHdl)->pictureHdl != NULL)
          fileType = kFileTypePICT;

        osError = FSpCreate(&fileSpec,kFileCreator,fileType,(*navReplyStruc).keyScript);
        if(osError != noErr)
        {
          NavDisposeReply(&(*navReplyStruc));
          return osError;
        }
      }

      // ..................... assign FSSpec to fileFSSpec field of window's document
structure

      (*docStrucHdl)->fileFSSpec = fileSpec;

      // ..................................... if file currently exists for document,
close it

      if((*docStrucHdl)->fileRefNum != 0)
      {
        osError = FSClose((*docStrucHdl)->fileRefNum);
        (*docStrucHdl)->fileRefNum = 0;
      }

      // ............................................... open file's data fork and
write file

      if(osError == noErr)
        osError = FSpOpenDF(&(*docStrucHdl)->fileFSSpec,fsRdWrPerm,&fileRefNum);

      if(osError == noErr)
      {
        (*docStrucHdl)->fileRefNum = fileRefNum;
        SetWTitle(windowRef,fileSpec.name);

        // ... ... ... ... ... ... ... ... ... ... ... proxy icon and file synchronisation
stuff

        SetPortWindowPort
(windowRef);                                                    /////
        SetWindowProxyFSSpec(windowRef,
&fileSpec);                                          /////
        GetWindowProxyAlias(windowRef,&((*docStrucHdl)-
>aliasHdl));                        /////
        SetWindowModified(windowRef,
false);                                                 /////
        // ... ... ... ... ... ... ... ... ... ... ... ... ... ...  write file using safe
save

        osError = doWriteFile(windowRef);

        NavCompleteSave(&(*navReplyStruc),kNavTranslateInPlace);
      }
    }
  }

  SetPortWindowPort(windowRef);
  GetWindowPortBounds(windowRef,&portRect);
  EraseRect(&portRect);
```

```
  InvalWindowRect(windowRef,&portRect);

  return osError;
}

// ************************************************************************
doSaveUsingFSRef

OSErr  doSaveUsingFSRef(WindowRef windowRef,NavReplyRecord *navReplyStruc)
{
  OSErr              osError = noErr;
  AEDesc             aeDesc;
  Size               dataSize;
  FSRef              fsRefParent, fsRefDelete;
  UniCharCount       nameLength;
  UniChar            *nameBuffer;
  FSSpec             fileSpec;
  docStructureHandle docStrucHdl;
  FInfo              fileInfo;
  SInt16             fileRefNum;
  Rect               portRect;

  osError = AECoerceDesc(&(*navReplyStruc).selection,typeFSRef,&aeDesc);
  if(osError == noErr)
  {
    // ..................................................................... get
FSRef

    dataSize = AEGetDescDataSize(&aeDesc);
    if(dataSize > 0)
      osError = AEGetDescData(&aeDesc,&fsRefParent,sizeof(FSRef));
    if(osError == noErr)
    {
      // ............................ get file name from saveFileName field of
NavReplyRecord

      nameLength = (UniCharCount) CFStringGetLength((*navReplyStruc).saveFileName);
      nameBuffer = (UniChar *) NewPtr(nameLength);
      CFStringGetCharacters((*navReplyStruc).saveFileName,CFRangeMake(0,nameLength),
                            &nameBuffer[0]);
      if(nameBuffer != NULL)
      {
        // ................................. if replacing, delete the file being replaced

        if((*navReplyStruc).replacing)
        {
          osError = FSMakeFSRefUnicode(&fsRefParent,nameLength,nameBuffer,
                                       kTextEncodingUnicodeDefault,&fsRefDelete);
          {
            if(osError == noErr)
              osError = FSDeleteObject(&fsRefDelete);
            if(osError == fBsyErr)
            {
              DisposePtr((Ptr) nameBuffer);
              return osError;
            }
          }
        }

        // .............. create file with Unicode name (but it can be written with an
FSSpec)

        if(osError == noErr)
        {
          osError = FSCreateFileUnicode(&fsRefParent,nameLength,nameBuffer,kFSCatInfoNone,
                                        NULL,NULL,&fileSpec);
```

```
          if(osError == noErr)
          {
            docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

            osError = FSpGetFInfo(&fileSpec,&fileInfo);

            if((*docStrucHdl)->editStrucHdl != NULL)
              fileInfo.fdType = kFileTypeTEXT;
            else if((*docStrucHdl)->pictureHdl != NULL)
              fileInfo.fdType = kFileTypePICT;
            fileInfo.fdCreator = kFileCreator;

            if(osError == noErr)
              osError = FSpSetFInfo(&fileSpec,&fileInfo);

            (*docStrucHdl)->fileFSSpec = fileSpec;

            // ..................................... open file's data fork and write
file

            if(osError == noErr)
              osError = FSpOpenDF(&fileSpec,fsRdWrPerm,&fileRefNum);

            if(osError == noErr)
            {
              (*docStrucHdl)->fileRefNum = fileRefNum;
              SetWTitle(windowRef,fileSpec.name);
              // ... ... ... ... ... ... ... ... ... ... proxy icon and file
synchronisation stuff

              SetPortWindowPort
(windowRef);                                                  /////
              SetWindowProxyFSSpec(windowRef,
&fileSpec);                                  /////
              GetWindowProxyAlias(windowRef,&((*docStrucHdl)-
>aliasHdl));                 /////
              SetWindowModified(windowRef,
false);                                       /////

              // ... ... ... ... ... ... ... ... ... ... ... ...  write file using safe save

              osError = doWriteFile(windowRef);

              NavCompleteSave(&(*navReplyStruc),kNavTranslateInPlace);
            }
          }
        }
      }

      DisposePtr((Ptr) nameBuffer);
    }

    AEDisposeDesc(&aeDesc);
  }

  SetPortWindowPort(windowRef);
  GetWindowPortBounds(windowRef,&portRect);
  EraseRect(&portRect);
  InvalWindowRect(windowRef,&portRect);

  return osError;
}

// ****************************************************************************
doWriteFile
```

```
OSErr  doWriteFile(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  FSSpec             fileSpecActual, fileSpecTemp;
  UInt32             currentTime;
  Str255             tempFileName;
  SInt16             tempFileVolNum, tempFileRefNum;
  SInt32             tempFileDirID;
  OSErr              osError = noErr;

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  fileSpecActual = (*docStrucHdl)->fileFSSpec;

  GetDateTime(&currentTime);
  NumToString((SInt32) currentTime,tempFileName);

  osError = FindFolder(fileSpecActual.vRefNum,kTemporaryFolderType,kCreateFolder,
                       &tempFileVolNum,&tempFileDirID);

  if(osError == noErr)
    osError = FSMakeFSSpec(tempFileVolNum,tempFileDirID,tempFileName,&fileSpecTemp);
  if(osError == noErr || osError == fnfErr)
    osError = FSpCreate(&fileSpecTemp,'trsh','trsh',smSystemScript);
  if(osError == noErr)
    osError = FSpOpenDF(&fileSpecTemp,fsRdWrPerm,&tempFileRefNum);
  if(osError == noErr)
  {
    if((*docStrucHdl)->editStrucHdl)
      osError = doWriteTextData(windowRef,tempFileRefNum);
    else if((*docStrucHdl)->pictureHdl)
      osError = doWritePictData(windowRef,tempFileRefNum);
  }
  if(osError == noErr)
    osError = FSClose(tempFileRefNum);
  if(osError == noErr)
    osError = FSClose((*docStrucHdl)->fileRefNum);
  if(osError == noErr)
    osError = FSpExchangeFiles(&fileSpecTemp,&fileSpecActual);
  if(osError == noErr)
    osError = FSpDelete(&fileSpecTemp);
  if(osError == noErr)
    osError = FSpOpenDF(&fileSpecActual,fsRdWrPerm,&(*docStrucHdl)->fileRefNum);

  if(osError == noErr)
    osError = doCopyResources(windowRef);

  return osError;
}

// ****************************************************************************
doReadTextFile

OSErr  doReadTextFile(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  SInt16             fileRefNum;
  TEHandle           textEditHdl;
  SInt32             numberOfBytes;
  Handle             textBuffer;
  OSErr              osError = noErr;

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  fileRefNum = (*docStrucHdl)->fileRefNum;

  textEditHdl = (*docStrucHdl)->editStrucHdl;
  (*textEditHdl)->txSize = 10;
```

```
  (*textEditHdl)->lineHeight = 15;

  SetFPos(fileRefNum,fsFromStart,0);
  GetEOF(fileRefNum,&numberOfBytes);

  if(numberOfBytes > 32767)
    numberOfBytes = 32767;

  if(!(textBuffer = NewHandle((Size) numberOfBytes)))
    return MemError();
  osError = FSRead(fileRefNum,&numberOfBytes,*textBuffer);
  if(osError == noErr || osError == eofErr)
  {
    HLockHi(textBuffer);
    TESetText(*textBuffer,numberOfBytes,(*docStrucHdl)->editStrucHdl);
    HUnlock(textBuffer);
    DisposeHandle(textBuffer);
  }
  else
    return osError;

  return noErr;
}

// ************************************************************************
doReadPictFile

OSErr  doReadPictFile(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  SInt16             fileRefNum;
  SInt32             numberOfBytes;
  OSErr              osError = noErr;

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  fileRefNum = (*docStrucHdl)->fileRefNum;

  GetEOF(fileRefNum,&numberOfBytes);
  SetFPos(fileRefNum,fsFromStart,512);
  numberOfBytes -= 512;

  if(!((*docStrucHdl)->pictureHdl = (PicHandle) NewHandle(numberOfBytes)))
    return MemError();

  osError = FSRead(fileRefNum,&numberOfBytes,*(*docStrucHdl)->pictureHdl);
  if(osError == noErr || osError == eofErr)
    return(noErr);
  else
    return osError;
}

// ************************************************************************
doWriteTextData

OSErr  doWriteTextData(WindowRef windowRef,SInt16 tempFileRefNum)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditHdl;
  Handle             editText;
  SInt32             numberOfBytes;
  SInt16             volRefNum;
  OSErr              osError = noErr;

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  textEditHdl = (*docStrucHdl)->editStrucHdl;
  editText = (*textEditHdl)->hText;
```

```
  numberOfBytes = (*textEditHdl)->teLength;
  osError = SetFPos(tempFileRefNum,fsFromStart,0);
  if(osError == noErr)
    osError = FSWrite(tempFileRefNum,&numberOfBytes,*editText);
  if(osError == noErr)
    osError = SetEOF(tempFileRefNum,numberOfBytes);
  if(osError == noErr)
    osError = GetVRefNum(tempFileRefNum,&volRefNum);
  if(osError == noErr)
    osError = FlushVol(NULL,volRefNum);

  if(osError == noErr)
    (*docStrucHdl)->windowTouched = false;

  return osError;
}

// ************************************************************************
doWritePictData

OSErr  doWritePictData(WindowRef windowRef,SInt16 tempFileRefNum)
{

  docStructureHandle docStrucHdl;
  PicHandle          pictureHdl;
  SInt32             numberOfBytes, dummyData;
  SInt16             volRefNum;
  OSErr              osError = noErr;

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  pictureHdl = (*docStrucHdl)->pictureHdl;

  numberOfBytes = 512;
  dummyData = 0;

  osError = SetFPos(tempFileRefNum,fsFromStart,0);

  if(osError == noErr)
    osError = FSWrite(tempFileRefNum,&numberOfBytes,&dummyData);

  numberOfBytes = GetHandleSize((Handle) (*docStrucHdl)->pictureHdl);

  if(osError == noErr)
  {
    HLock((Handle) (*docStrucHdl)->pictureHdl);
    osError = FSWrite(tempFileRefNum,&numberOfBytes,*(*docStrucHdl)->pictureHdl);
    HUnlock((Handle) (*docStrucHdl)->pictureHdl);
  }

  if(osError == noErr)
    osError = SetEOF(tempFileRefNum,512 + numberOfBytes);
  if(osError == noErr)
    osError = GetVRefNum(tempFileRefNum,&volRefNum);
  if(osError == noErr)
    osError = FlushVol(NULL,volRefNum);
  if(osError == noErr)
    (*docStrucHdl)->windowTouched = false;

  return osError;
}

// ************************************************************
getFilePutFileEventFunction

void  getFilePutFileEventFunction(NavEventCallbackMessage callBackSelector,
                                  NavCBRecPtr callBackParms,NavCallBackUserData callBackUD)
```

```
{
  OSErr              osError = noErr;
  NavReplyRecord     navReplyStruc;
  NavUserAction      navUserAction;
  SInt32             count, index;
  AEKeyword          theKeyword;
  DescType           actualType;
  FSSpec             fileSpec;
  Size               actualSize;
  FInfo              fileInfo;
  OSType             documentType;
  WindowRef          windowRef;
  AEDesc             aeDesc;
  AEKeyword          keyWord;
  DescType           typeCode;
  Rect               theRect;
  Str255             theString, numberString;
  docStructureHandle docStrucHdl;

  switch(callBackSelector)
  {
    case kNavCBUserAction:
      osError = NavDialogGetReply(callBackParms->context,&navReplyStruc);
      if(osError == noErr && navReplyStruc.validRecord)
      {
        navUserAction = NavDialogGetUserAction(callBackParms->context);

        switch(navUserAction)
        {
          // ..................................... click on Open button in Open
dialog

          case kNavUserActionOpen:
            if(gModalToApplicationNavDialogRef != NULL)
            {
              osError = AECountItems(&(navReplyStruc.selection),&count);
              if(osError == noErr)
              {
                for(index=1;index<=count;index++)
                {
                  osError = AEGetNthPtr(&(navReplyStruc.selection),index,typeFSS,
                                        &theKeyword,&actualType,&fileSpec,sizeof(fileSpec),
                                        &actualSize);
                  if((osError = FSpGetFInfo(&fileSpec,&fileInfo)) == noErr)
                  {
                    documentType = fileInfo.fdType;
                    osError = doOpenFile(fileSpec,documentType);
                    if(osError != noErr)
                      doErrorAlert(osError);
                  }
                }
              }
            }
            break;

          // ..................................... click on Save button in Save
Location dialog

          case kNavUserActionSaveAs:
            windowRef = callBackUD;
            osError = AECoerceDesc(&navReplyStruc.selection,typeFSRef,&aeDesc);
            if(osError == noErr)
            {
              osError = doSaveUsingFSRef(windowRef,&navReplyStruc);
              if(osError != noErr)
                doErrorAlert(osError);
```

```
            AEDisposeDesc(&aeDesc);
          }
          else
          {
            osError = doSaveUsingFSSpec(windowRef,&navReplyStruc);
            if(osError != noErr)
              doErrorAlert(osError);
          }
          break;

        // ............................. click on Choose button in Choose a Folder
dialog

        case kNavUserActionChoose:
          if((osError = AEGetNthPtr(&(navReplyStruc.selection),1,typeFSS,&keyWord,
&typeCode,
                                    &fileSpec,sizeof(FSSpec),&actualSize)) == noErr)
          {
            FSMakeFSSpec(fileSpec.vRefNum,fileSpec.parID,fileSpec.name,&fileSpec);
          }
          windowRef = callBackUD;
          SetPortWindowPort(windowRef);
          TextSize(10);
          SetRect(&theRect,0,271,600,300);
          EraseRect(&theRect);
          doCopyPString(fileSpec.name,theString);
          doConcatPStrings(theString, "\p   Volume Reference Number: ");
          NumToString((SInt32) fileSpec.vRefNum,numberString);
          doConcatPStrings(theString,numberString);
          doConcatPStrings(theString, "\p   Parent Directory ID: ");
          NumToString((SInt32) fileSpec.parID,numberString);
          doConcatPStrings(theString,numberString);
          MoveTo(10,290);
          DrawString(theString);
          break;
      }
      osError = NavDisposeReply(&navReplyStruc);
    }
    break;

  case kNavCBTerminate:
    if(gModalToApplicationNavDialogRef != NULL)
    {
      NavDialogDispose(gModalToApplicationNavDialogRef);
      gModalToApplicationNavDialogRef = NULL;
    }
    else
    {
      windowRef = callBackUD;
      docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
      if((*docStrucHdl)->modalToWindowNavDialogRef != NULL)
      {
        NavDialogDispose((*docStrucHdl)->modalToWindowNavDialogRef);
        (*docStrucHdl)->modalToWindowNavDialogRef = NULL;
      }
    }
    break;
  }
}

// ************************************************************
askSaveDiscardEventFunction

void askSaveDiscardEventFunction(NavEventCallbackMessage callBackSelector,
                                 NavCBRecPtr callBackParms,NavCallBackUserData callBackUD)
{
```

```
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  NavUserAction      navUserAction;
  OSErr              osError = noErr;
  Rect               portRect;

  switch(callBackSelector)
  {
    case kNavCBUserAction:
      windowRef = callBackUD;
      docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

      if((*docStrucHdl)->modalToWindowNavDialogRef != NULL)
      {
        navUserAction = NavDialogGetUserAction(callBackParms->context);
        switch(navUserAction)
        {
          // ................................. click on Save button in Save Changes alert

          case kNavUserActionSaveChanges:
            osError = doSaveCommand();
            if(osError != noErr)
              doErrorAlert(osError);
          // ................................. click on Don't Save button in Save Changes
alert

          case kNavUserActionDontSaveChanges:
            NavDialogDispose((*docStrucHdl)->modalToWindowNavDialogRef);
            if(gRunningOnX)
            {
              osError = doCloseDocWindow(windowRef);
              if(osError != noErr)
                doErrorAlert(osError);
            }
            else
              gCloseDocWindow = true;
            break;

          // ..................................... click on OK button in Discard Changes
alert

          case kNavUserActionSaveChanges:
            osError = doSaveCommand();
            if(osError != noErr)
              doErrorAlert(osError);

          // ................................. click on Don't Save button in Save Changes
alert

          case kNavUserActionDontSaveChanges:
            NavDialogDispose((*docStrucHdl)->modalToWindowNavDialogRef);
            osError = doCloseDocWindow(windowRef);
            if(osError != noErr)
              doErrorAlert(osError);
            break;

          // ..................................... click on OK button in Discard Changes
alert

          case kNavUserActionDiscardChanges:
            GetWindowPortBounds(windowRef,&portRect);
            SetPortWindowPort(windowRef);
            EraseRect(&portRect);

            if((*docStrucHdl)->editStrucHdl != NULL && (*docStrucHdl)->fileRefNum != 0)
            {
```

```
              osError = doReadTextFile(windowRef);
              if(osError != noErr)
                doErrorAlert(osError);
            }
            else if((*docStrucHdl)->pictureHdl != NULL)
            {
              KillPicture((*docStrucHdl)->pictureHdl);
              (*docStrucHdl)->pictureHdl = NULL;

              osError = doReadPictFile(windowRef);
              if(osError != noErr)
                doErrorAlert(osError);
            }
            (*docStrucHdl)->windowTouched = false;
            SetWindowModified(windowRef,
false);                                          /////
            InvalWindowRect(windowRef,&portRect);

            NavDialogDispose((*docStrucHdl)->modalToWindowNavDialogRef);
            (*docStrucHdl)->modalToWindowNavDialogRef = NULL;
            break;

          // ................. click on Cancel button in Save Changes or Discard Changes
alert

          case kNavUserActionCancel:
            if((*docStrucHdl)->isAskSaveChangesDialog == true)
            {
              gQuittingApplication = false;
              (*docStrucHdl)->isAskSaveChangesDialog = false;
            }
            NavDialogDispose((*docStrucHdl)->modalToWindowNavDialogRef);
            (*docStrucHdl)->modalToWindowNavDialogRef = NULL;
            break;
        }
      }
      break;
  }
}

// ****************************************************************************
doCopyResources

OSErr  doCopyResources(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  OSType             fileType;
  OSErr              osError = noErr;
  SInt16             fileRefNum;
  Handle             editTextHdl, textResourceHdl;

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

  if((*docStrucHdl)->editStrucHdl)
    fileType = kFileTypeTEXT;
  else if((*docStrucHdl)->pictureHdl)
    fileType = kFileTypePICT;

  FSpCreateResFile(&(*docStrucHdl)->fileFSSpec,kFileCreator,fileType,smSystemScript);

  osError = ResError();
  if(osError == noErr)
    fileRefNum = FSpOpenResFile(&(*docStrucHdl)->fileFSSpec,fsRdWrPerm);

  if(fileRefNum > 0)
  {
```

```
    osError = doCopyAResource('STR ',-16396,gAppResFileRefNum,fileRefNum);
    if(fileType == kFileTypePICT)
    {
      doCopyAResource('pnot',128,gAppResFileRefNum,fileRefNum);
      doCopyAResource('PICT',128,gAppResFileRefNum,fileRefNum);
    }

    if(!gRunningOnX && fileType == kFileTypeTEXT)
    {
      doCopyAResource('pnot',129,gAppResFileRefNum,fileRefNum);

      editTextHdl = (*(*docStrucHdl)->editStrucHdl)->hText;
      textResourceHdl = NewHandleClear(1024);
      BlockMoveData(*editTextHdl,*textResourceHdl,1024);
      UseResFile(fileRefNum);
      AddResource(textResourceHdl,'TEXT',129,"\p");
      if(ResError() == noErr)
        UpdateResFile(fileRefNum);
      ReleaseResource(textResourceHdl);
    }
  }
  else
    osError = ResError();

  if(osError == noErr)
    CloseResFile(fileRefNum);

  osError = ResError();
  return osError;
}

// ************************************************************************
doCopyAResource

OSErr  doCopyAResource(ResType resourceType,SInt16 resourceID,SInt16 sourceFileRefNum,
                       SInt16 destFileRefNum)
{
  Handle  sourceResourceHdl;
  Str255  sourceResourceName;
  ResType ignoredType;
  SInt16  ignoredID;

  UseResFile(sourceFileRefNum);

  sourceResourceHdl = GetResource(resourceType,resourceID);

  if(sourceResourceHdl != NULL)
  {
    GetResInfo(sourceResourceHdl,&ignoredID,&ignoredType,sourceResourceName);
    DetachResource(sourceResourceHdl);
    UseResFile(destFileRefNum);
    AddResource(sourceResourceHdl,resourceType,resourceID,sourceResourceName);
    if(ResError() == noErr)
      UpdateResFile(destFileRefNum);
  }
  ReleaseResource(sourceResourceHdl);

  return ResError();
}

//
****************************************************************************************
// SynchroniseFiles.c
//
****************************************************************************************
```

```
// ..........................................................................
includes

#include "Files.h"

// ..................................................................... global
variables

extern SInt16  gCurrentNumberOfWindows;

// ************************************************************************
doSynchroniseFiles

void  doSynchroniseFiles(void)
{
  WindowRef          windowRef;
  SInt16             trashVRefNum;
  SInt32             trashDirID;
  docStructureHandle docStrucHdl;
  Boolean            aliasChanged;
  AliasHandle        aliasHdl;
  FSSpec             newFSSpec;
  OSErr              osError;

  windowRef = FrontNonFloatingWindow();

  while(windowRef != NULL)
  {
    docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

    if(docStrucHdl != NULL)
    {
      if((*docStrucHdl)->aliasHdl == NULL)
        break;

      aliasChanged = false;
      aliasHdl = (*docStrucHdl)->aliasHdl;
      ResolveAlias(NULL,aliasHdl,&newFSSpec,&aliasChanged);

      if(aliasChanged)
      {
        (*docStrucHdl)->fileFSSpec = newFSSpec;
        SetWTitle(windowRef,newFSSpec.name);
      }

      osError = FindFolder(kUserDomain,kTrashFolderType,kDontCreateFolder,&trashVRefNum,
                           &trashDirID);
      if(osError == noErr)
      {
        do
        {
          if(newFSSpec.parID == fsRtParID)
            break;

          if((newFSSpec.vRefNum == trashVRefNum) && (newFSSpec.parID == trashDirID))
          {
            FSClose((*docStrucHdl)->fileRefNum);
            if((*docStrucHdl)->editStrucHdl)
              TEDispose((*docStrucHdl)->editStrucHdl);
            if((*docStrucHdl)->pictureHdl)
              KillPicture((*docStrucHdl)->pictureHdl);
            DisposeHandle((Handle) docStrucHdl);
            DisposeWindow(windowRef);
            gCurrentNumberOfWindows --;
            break;
          }
```

```
        }  while(FSMakeFSSpec(newFSSpec.vRefNum,newFSSpec.parID,"\p",&newFSSpec) == noErr);
      }
    }

    windowRef = GetNextWindow(windowRef);
  }
}

//
************************************************************************************************
// ChooseAFolderDialog.c
//
************************************************************************************************

// ..........................................................................
includes

#include "Files.h"

// ..................................................................... global
variables

extern NavEventUPP  gGetFilePutFileEventFunctionUPP ;
extern NavDialogRef gModalToApplicationNavDialogRef;

// ********************************************************************
doChooseAFolderDialog

OSErr  doChooseAFolderDialog(void)
{
  OSErr                    osError = noErr;
  NavDialogCreationOptions dialogOptions;
  WindowRef                windowRef, parentWindowRef;
  Str255                   message;

  osError = NavGetDefaultDialogCreationOptions(&dialogOptions);
  if(osError == noErr)
  {
    if((osError = GetSheetWindowParent(FrontWindow(),&parentWindowRef)) == noErr)
      windowRef = parentWindowRef;
    else
      windowRef = FrontWindow();
    GetIndString(message,rMiscStrings,sChooseAFolder);
    dialogOptions.message = CFStringCreateWithPascalString(NULL,message,
                                                           CFStringGetSystemEncoding());
    dialogOptions.modality = kWindowModalityAppModal;

    osError = NavCreateChooseFolderDialog(&dialogOptions,gGetFilePutFileEventFunctionUPP ,
                                          NULL,windowRef,&gModalToApplicationNavDialogRef);

    if(osError == noErr && gModalToApplicationNavDialogRef != NULL)
    {
      osError = NavDialogRun(gModalToApplicationNavDialogRef);
      if(osError != noErr)
      {
        NavDialogDispose(gModalToApplicationNavDialogRef);
        gModalToApplicationNavDialogRef = NULL;
      }
    }
  }

  return osError;
}

//
************************************************************************************************
```

Demonstration Program Files Comments

When the program is run, the user should:

- Exercise the File menu by opening the supplied TEXT and PICT files, saving those files, saving those files under new names, closing files, opening the new files, attempting to open files that are already open, attempting to save files to new files with existing names, making open windows "touched" by choosing the first item in the Demonstration menu, reverting to the saved versions of files associated with "touched" windows, choosing Quit when one "touched" window is open, choosing Quit when two or more "touched" windows are open, and so on.

- Choose, via the Show pop-up menu button, the file types required to be displayed in the Open dialog.

- Choose the Choose a Folder item from the Demonstration menu to display the Choose a Folder dialog, and choose a folder using the Choose button at the bottom of the dialog. (The name of the chosen folder will be drawn in the bottom-left corner of the front window.)

- With either the PICT Document or the TEXT Document open

    - With the document's Finder icon visible, drag the window proxy icon to the desktop or to another open folder, noting that the Finder icon moves to the latter. Then choose Touch Window from the Demonstration menu to simulate unsaved changes to the document. Note that the proxy icon changes to the disabled state. Then save the file, proving the correct operation of the file synchronization function. Note that, after the save, the window proxy icon changes back to the enabled state.

    - Command-click the window's title to display the window path pop-up menu, choose a folder from the menu, and note that the Finder is brought to the foreground and the chosen folder opens.

The program may be run from within CodeWarrior to demonstrate responses to the File menu commands and the Choose a Folder dialog.

The built application, together with the supplied 'TEXT' and 'PICT' files, may be used to demonstrate the additional aspect of integrating the receipt of required Apple events with the overall file handling mechanism. To prove the correct handling of the required Apple events, the user should:

- Open the application by double-clicking the application icon, noting that a new document window is opened after the application is launched and the Open Application event is received.

- Double click on a document icon, or select one or more document icons and either drag those icons to the application icon or choose Open from the Finder's File menu, noting that the application is launched and the selected files are opened when the Open Documents event is received.

- Close all windows and double-click the application icon, noting that the application responds to the Re-open Application event by opening a new window.

- With the PICT Document and the TEXT Document open and "touched," and several other windows open, choose Restart or Shut Down from the Mac OS 8/9 Finder's Special menu or the Mac OS X Apple menu (thus invoking a Quit Application event), noting that, for "touched" windows, the Save Changes alert is presented asking the user whether the file should be saved before the shutdown process proceeds. (On Mac OS X, a Review Unsaved alert will be presented at first.)

Note, however, that no printing functions are included. Thus, selecting one or more document icons and choosing Print from the Finder's File menu (Mac OS 8/9) will result in the file opening but not printing.

Files.h

defines

Constants are established for a 'STR#' resource containing error strings for three specific error conditions, a 'STR#' resource containing the application's name and the message string for the Choose a Folder dialog, and the 'open' resource containing the file types list.

KFileCreator represents the application's signature and the next two constants represent the file types that are readable and writeable by the application.

typedefs

Each window created by the program will have an associated document structure. The docStructure data type will be used for document structures.

The editStrucHdl field will be assigned a handle to a TextEdit structure ('TEXT' files). The pictureHdl field will be assigned a handle to a Picture structure ('PICT' files). The fileRefNum and fileFSSpec fields will be assigned the file reference number and the file system specification structure of the file associated with the window. When a file is opened, the aliasHdl field will be assigned a handle to a structure of type AliasRecord, which contains the alias data for the file. The windowTouched field will be set to true when a window has been made "touched."

When modal-to-the-window Navigation Services dialogs (Save Location, Save Changes, and Discard Changes alerts) are created, the dialog reference will be assigned to the modalToWindowNavDialogRef field. When Save Changes and Discard Changes alerts are created, a universal procedure pointer to the associated event (callback) function will be assigned to the askSaveDiscardChangesDialog field. When a Save Changes alert is created, the isAskSaveChangesDialog field will be set to true to enable the associated event (callback) function to reset a "quitting application" flag if the user clicks the Cancel button in a Save Changes alert (but not if the user clicks the Cancel button in a Discard Changes alert).

Files.c

Global Variables

gAppResFileRefNum will be assigned the file reference number of the application's resource fork. gGetFilePutFileEventFunctionUPP will be assigned a universal procedure pointer to the event (callback) function associated with the Open, Save Location, and Choose a Folder dialogs. gQuittingApplication is set to true in certain circumstances within quitAppEventHandler and to false if the Cancel button is clicked in a Save Changes or Review Unsaved alert.

main

The file reference number of the application's resource fork (which is opened automatically at application launch) is assigned to the global variable gAppResFileRefNum.

After the required Apple event handlers are installed, the program's application event handler and a timer are installed. The timer is set to fire at an interval of 15 ticks, and will be used to trigger calls to the function doIdle, which calls the program's file synchronization function.

A universal procedure pointer to the event (callback) function associated with the Open, Save Location, and Choose a Folder dialogs is created and assigned to the global variable gGetFilePutFileEventFunctionUPP.

doInstallAEHandlers

doInstallAEHandlers installs handlers for the Open Application, Re-Open Application, Open Document, Print Documents, and Quit Application events. Since the program installs its own Quit Application event handler, the default Quit Application event handler will not be installed when RunApplicationEventLoop is called.

windowEventHandler

`windowEventHandler` is the program's window event handler (a callback function), which is installed on all document windows.

Note that, when the event type `kEventWindowClose` is received, the constant passed in the call to `doCloseCommand` depends on whether the global variable `gQuittingApplication` is set to true or false. This constant affects the text in the Save Changes alert.

Note also that no code is required in a Carbon application to handle window path pop-up menus. The standard window handler handles all user interaction with window path pop-up menus, including bringing the Finder to the front when the user chooses a folder.

doIdle

`doIdle` is called when the installed timer fires. If the front window is a document window, `doSynchroniseFiles` is called to synchronizes the application with the actual current location (and name) of its currently open document files.

doDrawContent

`doDrawContent` is called when the `kEventWindowDrawContent` event type is received. It performs such window updating as is necessary for the satisfactory execution of the demonstration aspects of the program.

doMenuChoice

At the File_Close case, `kNavSaveChangesClosingDocument` is passed in the call to `doCloseCommand`. This affects the wording in the Save Changes alert.

doAdjustMenus

If the program is running on Mac OS X, `GetSheetWindowParent` is called as a way of determining whether the frontmost window is a sheet. If it is, the File and Demonstration menus are adjusted accordingly.

doTouchWindow

`doTouchWindow` is called when the user chooses the Touch Window item in the Demonstration menu. Changing the content of the in-memory version of a file is only simulated in this program. The text "WINDOW TOUCHED" is drawn in window and the windowTouched field of the document structure is set to true.

`SetWindowModified` is called with true passed in the modified parameter. This causes the proxy icon to appear in the disabled state, indicating that the window has unsaved changes.

openAppEventHandler, reopenAppEventHandler, and openAndPrintDocsEventHandler

The handlers for the first four required Apple events are essentially identical to those in the demonstration program AppleEvents. One major difference is that one handler (`openAndPrintDocsEventHandler`) is used for both the Open Documents and Print Documents events, with a value passed in the handler's `handlerRefcon` parameter advising the handler which of the two events has been received.

Most programs should simply open a new untitled window on receipt of an Open Application event. Accordingly, `openAppEventHandler` simply calls the same function (`doNewCommand`) as is called when the user chooses New from the File menu.

On receipt of a Re-Open Application event, if no windows are currently open, `doNewCommand` is called to open a window.

The demonstration program supports both `'TEXT'` and `'PICT'` files. On receipt of an Open Application event, it is thus

necessary to determine the type of each file specified in the event. Accordingly, within `openAndPrintDocsEventHandler`, the call to `FSpGetFInfo` returns the Finder information from the volume catalog entry for the file relating to the specified `FSSpec` structure. The `fdType` field of the `FInfo` structure "filled-in" by `FSpGetFInfo` contains the file type. This, together with the `FSSpec` structure, is then passed in the call to `doOpenFile`. (`doOpenFile` is also called when the user chooses Open from the File menu.)

quitAppEventHandler

Much of the code in `quitAppEventHandler` has to do with the requirement, on Mac OS X only, to present a Review Unsaved alert if more than one window with unsaved changes is open when the event is received.

If no windows are open, `QuitApplicationEventLoop` is called to close the application down. If at least one window is open, the following occurs.

`GetFrontWindowOfClass` is called to determine whether any window has a sheet. If so, that window is brought to the front and activated and the handler returns immediately, keeping the application alive.

The do-while loop walks the window list counting the number of document windows with unsaved changes (that is, "touched" windows) and, at the same time, bringing those windows to the front. At the next block, if there are no touched document windows, `QuitApplicationEventLoop` is called to close the application down.

If the application is running on Mac OS X, the following occurs:

- If there is only one touched window open, the flag `gQuittingApplication` is set to true and a `kEventWindowClose` event is created and sent to the front window. As will be see, this results in a sequence involving `doCloseCommand` and `doCloseDocWindow` whereby all untouched windows in front of the touched window are disposed of and a Save Changes alert is presented for the touched window. In this sequence, if the event handler for the Save Changes alert detects a Cancel button click, `gQuittingApplication` will be set to false, an action which will cause the process of closing down the remaining windows and the application to be terminated. If the Save or Don't Save buttons are clicked, all remaining windows will be closed down, and the program will be closed down by a call to `QuitApplicationEventLoop`, within the function `doCloseDocWindow`.
- If more than one window has been touched, `doReviewChangesAlert` is called to create, display and handle a Review Changes alert. If the Review Changes button is hit, the flag `gQuittingApplication` is set to true and a `kEventWindowClose` event is created and sent to the front window, resulting in the same general process of close-down, and possible termination of that close-down process, described above. If the Cancel button is hit, the flag `gQuittingApplication` is set to false (which defeats the execution of the last block of code in `doCloseDocWindow`) and `quitAppEventHandler` simply returns. If the Discard Changes button is hit, `QuitApplicationEventLoop` is called to terminate the program.

If the application is running on Mac OS 8/9, a Review Unsaved alert is not invoked. Instead, a `kEventWindowClose` event is created and sent to the front window. This results in the same general process of close-down, and possible termination of that close-down process, described above. If the Cancel button is not clicked in all Save Changes alerts, all windows will be closed down, and `QuitApplicationEventLoop` called, within the function `doCloseDocWindow`.

NewOpenCloseSave.c

Global Variables

`gModalToApplicationNavDialogRef` will be assigned the dialog reference for the Open File dialog, which is made application-modal. `gCurrentNumberOfWindows` keeps a count of the number of windows opened. `gDestRect` and `gViewRect` are used to set the destination and view rectangles for the TextEdit structures associated with `'TEXT'` files.

doNewCommand

`doNewCommand` is called when the user chooses New from the File menu and when an Open Application or Re-Open Application event is received.

Since this demonstration does not support the actual entry of text or the drawing of graphics, the document type passed to `doNewDocWindow` is immaterial. The document type `'TEXT'` is passed in this instance simply to keep `doNewDocWindow` happy.

If `doNewDocWindow` returns no error, `SetWindowProxyCreatorAndType` is called to set the proxy icon for the window. (A new, untitled window, even though it has no associated file, needs a proxy icon to maintain visual consistency with other windows which have associated files.) The proxy icon will display in the disabled state, indicating, in this particular case, that the window has no associated file rather than unsaved changes.

The creator code and file type passed in the second and third parameters of `SetWindowProxyCreatorAndType` determine the icon to be displayed.)

doOpenCommand

`doOpenCommand`, which is called when the user chooses Open from the File menu, uses Navigation Services 3.0 functions to create and display a application-modal Open dialog.

`NavGetDefaultDialogCreationOptions` initializes the specified `NavDialogCreationOptions` structure with the defaults.

`GetIndString` retrieves the application's name and assigns it to an `Str255` variable. This is then converted to a `CFString` and assigned to the `clientName` field of the `NavDialogCreationOptions` structure. This will cause the application's name to appear in the dialog's title bar.

The next line assigns a value to the modality field of the `NavDialogCreationOptions` structure,which will cause the dialog to be application-modal.

An `'open'` resource containing the file type list is then read in and the handle assigned a variable of type `NavTypeListHandle`. (The `'open'` resource specifies that `'TEXT'` and `'PICT'` file types are supported.)

The call to `NavCreateGetFileDialog` creates the dialog. Since the default options are being used, multiple file selection is allowed. A universal procedure pointer to the event function `getFilePutFileEventFunction`, which will respond to button clicks in the dialog, is passed in the third parameter. No preview function or filter function is used, and no user data is passed in. The last parameter (a global variable) receives the dialog reference.

The call to `NavDialogRun` displays the dialog.

doCloseCommand

`doCloseCommand` is called when the user chooses Close from the File menu or clicks in the window's go-away box. It is also called successively for each open window when a Quit Application Apple event is received.

The first two lines get a reference to the front window and establish whether the front window is a document window or a modeless dialog.

If the front window is a document window, the handle to the window's document structure is retrieved from the window's window object, allowing a check of whether the window is touched (that is, has unsaved changes). If it does, `doCreateAskSaveChangesDialog` is called to create and display a Save Changes alert and the function returns, otherwise `doCloseDocWindow` is called. Prior to the call to `doCreateAskSaveChangesDialog`, if the window is collapsed (Mac OS 8/9) or minimized in the dock (Mac OS X) it is first uncollapsed or brought out of the Dock.

No modeless dialogs are used by this program. However, if the front window was a modeless dialog, the appropriate action would be taken at the second case.

doSaveCommand

`doSaveCommand` is called when the user chooses Save from the File menu or clicks the Save button in a Save Changes alert.

The first two lines get the `WindowRef` for the front window and retrieve the handle to that window's document structure. If a file currently exists for the document in this window, the function `doWriteFile` is called. The next four lines are incidental to the demonstration; they simply remove the words "WINDOW TOUCHED" from the window.

`SetWindowModified` is called with false passed in the modified parameter. This causes the window proxy icon to appear in the enabled state, indicating no unsaved changes.

doSaveAsCommand

`doSaveAsCommand` uses Navigation Services 3.0 functions to create and display a window-modal Save Location dialog. It is called when the user chooses Save As from the File menu. It is also called by `doSaveCommand` if the user chooses Save when the front window contains a document for which no file currently exists.

`NavGetDefaultDialogCreationOptions` initializes the specified `NavDialogCreationOptions` structure with the defaults. The first line in the if block unsets the "allow saving of stationery files" bit (one of the defaults). On Mac OS 8/9, this means that the dialog will not contain the Format: pop-up menu.

`GetWTitle` gets the front window's title into an `Str255` variable. This is then converted to a `CFString` and assigned to the `saveFileName` field of the `NavDialogCreationOptions` structure. This will be the default name for the saved file and will appear in the Name (OS 8/9) and Save As (OS X) edit text fields in the Save Location dialog.

The next two lines assign the application's name to the `clientName` field of the `NavDialogCreationOptions` structure. This will then appear in the dialog's title bar.

The next two lines assign the window reference to the `parentWindow` field of the `NavDialogCreationOptions` structure and assign a value to the modality field which will cause the dialog to be window-modal.

The next block gets the file type from the window's document structure into a local variable.

The call to `NavCreatePutFileDialog` creates the dialog. A universal procedure pointer to the event function `getFilePutFileEventFunction`, which will respond to button clicks in the dialog, is passed in the fourth parameter. The window reference is passed in the fifth (user data) parameter. This will be passed to the event function. The dialog reference is assigned to a field of the window's document structure.

The call to `NavDialogRun` displays the dialog.

doRevertCommand

`doRevertCommand`, which is called when the user chooses Revert from the File menu, uses Navigation Services 3.0 functions to create and display a window-modal Discard Changes alert. The general approach is similar to that used to create and display the Save Location dialog, the main difference being that a universal procedure pointer to the event function `askSaveDiscardEventFunction` is stored in the `askSaveDiscardEventFunctionUPP` field of the window's document structure.

doNewDocWindow

`doNewDocWindow` is called by `doNewCommand` and `doOpenFile`.

If the current number of open windows is the maximum allowable by this program, the function immediately exits, returning an error code which will cause an advisory error alert to be displayed.

The call to `CreateNewWindow` creates a new window with the standard document window attributes and with the standard window event handler installed. `SetWTitle` is called to set the window's title to "untitled". `ChangeWindowAttributes` is called to remove the zoom box/button and grow box from the window. The call to `InstallWindowEventHandler` installs the program's window event handler on the window.

The call to `NewHandle` allocates memory for the window's document structure. If this call is not successful, the window is disposed of and the function returns with the error code returned by `MemError`. The call to `SetWRefCon` assigns the handle to the document structure to the window structure's `refCon` field. The next block initializes fields of the document structure.

If the document type is `'TEXT'`, the if block executes, creating a TextEdit structure and assigning a handle to that structure to the `editStrucHdl` field of the document structure. (Note that the processes here are not explained in detail because TextEdit and TextEdit structures are not central to the demonstration. For the purposes of the demonstration, it is sufficient to understand that the text data retrieved from, and saved to, disk is stored in a TextEdit structure. (TextEdit is addressed in detail in Chapter 21.))

If the Boolean value passed to `doNewDocWindow` was set to true, the call to `ShowWindow` makes the window visible, otherwise the window is left invisible. The penultimate line increments the global variable which keeps track of the number of open windows.

doCloseDocWindow

`doCloseDocWindow` is called from `doCloseCommand` when the subject window is not touched and from the Save Changes alert event handler `askSaveDiscardEventFunction` when the user clicks the Save or Don't Save buttons in a Save Changes alert.

The `FSClose` call closes the file, and `FlushVol` stores to disk all unwritten data currently in the volume buffer.

If the document is a text document, the TextEdit structure is disposed of. If it is a picture document, the Picture structure is disposed of. Finally, the document structure and window are disposed of and the global variable which keeps track of the number of open windows is decremented.

The last block executes only if `gQuittingApplication` has been set to true in the function `quitAppEventHandler`. If all windows have been closed, `QuitApplicationEventLoop` is called to terminate the program; otherwise a `kEventWindowClose` is created and sent to the front window, causing `doCloseCommand` to be called from the window event handler. This repetitive calling of `doCloseCommand` and `doCloseDocWindow` will continue until no windows remain or until `gQuittingApplication` is set to false by a click in the Cancel button in a Save Changes or, on Mac OS X only, a Review Unsaved alert.

doOpenFile

`doOpenFile` opens a new document window and calls the functions, which read in the file. It is called by the event function `getFilePutFileEventFunction` when an Open button click occurs in an Open dialog. The event function passes the file system specification structure and document type to `doOpenFile`.

The call to `doNewDocWindow` opens a new window and creates an associated document structure. `SetWTitle` sets the window's title using information in the file system specification structure. `FSpOpenDF` opens the file's data fork. If this call is not successful, the window is disposed of and the function returns. The next three lines assign the file reference number and file system specification structure to the relevant fields of the document structure.

The next block calls the appropriate function for reading in the file, depending on whether the file type is of type `'TEXT'` or `'PICT'`. If the file is read in successfully, `ShowWindow` makes the window visible.

Just before the call to `ShowWindow`, `SetWindowProxyFSSpec` is called to establish a proxy icon for the window and associate the file with the window. (The creator code and file type of the file determine the icon to be displayed.) `GetWindowProxyAlias` assigns a copy of the alias data for the file to the `aliasHdl` field of the window's document structure. (This is used by the file synchronisation function.) `SetWindowModified` is called with false passed in the modified parameter. This causes the window proxy icon to appear in the enabled state, indicating no unsaved changes.

doCreateAskSaveChangesDialog

`doCreateAskSaveChangesDialog`, which is called from `doCloseCommand`, uses Navigation Services 3.0 functions to create and display a window-modal Save Changes alert. The general approach is similar to that used to create and display

the Discard Changes alert, but note that in this case that the `isAskSaveChangesDialog` field of the window's document structure is set to true.

Note also that, if the program is running on Mac OS 8/9, and if `gCloseDocWindow` is true, `doCloseDocWindow` is called to close the file, flush the volume, and close down the window. (`gCloseDocWindow` is set to true in the callback function `askSaveDiscardEventFunction` if the user clicks the Don't Save push button button in the Save Changes alert.)

doSaveUsingFSSPec

As will be seen in the event function `getFilePutFileEventFunction`, when the user clicks on the Save button in a Save Location dialog, `AECoerceDesc` is called on the descriptor structure in the selection field of the `NavReplyRecord` structure in an attempt to coerce it to type `FSRef`. If the call is successful (meaning that the program is running on Mac OS X), `doSaveUsingFSRef` is called to perform the save using the HFS Plus API. If the call is not successful (meaning that the program is running on Mac OS 8/9) this function (`doSaveUsingFSSpec`) is called.

A descriptor structure is returned in the selection field of the `NavReplyRecord` structure. `AEGetNthPtr` coerces the descriptor structure to `typeFSS` and stores the result in the local variable `fileSpec`.

The name field of `fileSpec` will be empty at this stage. Accordingly, the Navigation Services 3.0 function `NavDialogGetSaveFileName` is called to get a `CFStringRef` to the filename from the dialog object, which is converted to a Pascal string and assigned to the name field of fileSpec.

If the value in the replacing field of the `NavReplyRecord` structure indicates that the file is not being replaced, `FSpCreate` is called to create a new file of the specified type and with the application's signature as the specified creator. If this call is not successful, the `NavReplyRecord` structure is disposed of and the function returns.

The file system specification structure returned by the `FSpCreate` call is assigned to the `fileFSSpec` field of the window's document structure. If a file currently exists for the document, that file is closed by the call to `FSClose`. The data fork of the newly created file is then opened by a call to `FSpOpenDF`, the `fileRefNum` field of the document structure is assigned the file reference number returned by `FSpOpenDF`, the window's title is set to the new file's name, and the function `doWriteFile` is called to write the document to the new file. `NavCompleteSave` is called to complete the save operation.

Just before the call to `doWriteFile`, `SetWindowProxyFSSpec` is called to establish a proxy icon for the window and associate the file with the window. (The creator code and file type of the file determine the icon to be displayed.) `GetWindowProxyAlias` assigns a copy of the alias data for the file to the `aliasHdl` field of the window's document structure. (This is used by the file synchronization function.) `SetWindowModified` is called with false passed in the modified parameter. This causes the window proxy icon to appear in the enabled state, indicating no unsaved changes.

doSaveUsingFSRef

`doSaveUsingFSRef`, which is called from the event function `getFilePutFileEventFunction`, performs the save using the HFS Plus API. The main if block executes only if the call to `AECoerceDesc` is successful in coercing the descriptor structure in the selection field of the `NavReplyRecord` to type `FSRef`.

In Carbon, the `dataHandle` field of descriptor structures is opaque. Thus `AEGetDescData` is used to extract the data in this field, which is assigned to the local variable `fsRefParent`. This is the `FSRef` for the parent directory.

At the next block, `CFStringGetLength` is called to get the number of 16-bit Unicode characters in the `saveFileName` field of the `NavReplyRecord` structure. This facilitates the call to `CFStringGetCharacters`, which extracts the contents of the string into a buffer.

If the value in the replacing field of the `NavReplyRecord` structure indicates that the file is being replaced, the existing file is first deleted. `FSMakeFSRefUnicode`, given a parent directory and Unicode file name, creates an `FSRef` for the file. This is passed in the call to `FSDeleteObject`, which deletes the file.

The call to `FSCreateFileUnicode` creates a new file with the Unicode name. On return, the last parameter contains a file

system specification structure for the new file. (Although the file is created with a Unicode name, it can be written using a file system specification structure.)

The call to FSpGetFInfo gets the Finder information from the volume catalog entry for the file. The file type extracted from the window's document structure is then assigned to the fdType field of the returned FInfo structure, following which FSpSetFInfo is called to set the new Finder information in the file's volume catalog entry.

The file system specification structure is assigned to the fileFSSpec field of the window's document structure.

The data fork of the newly created file is then opened by a call to FSpOpenDF, the fileRefNum field of the document structure is assigned the file reference number returned by FSpOpenDF, the window's title is set to the new file's name, and the function doWriteFile is called to write the document to the new file. NavCompleteSave is called to complete the save operation.

Just before the call to doWriteFile, SetWindowProxyFSSpec is called to establish a proxy icon for the window and associate the file with the window. (The creator code and file type of the file determine the icon to be displayed.) GetWindowProxyAlias assigns a copy of the alias data for the file to the aliasHdl field of the window's document structure. (This is used by the file synchronisation function.) SetWindowModified is called with false passed in the modified parameter. This causes the window proxy icon to appear in the enabled state, indicating no unsaved changes.

doWriteFile

doWriteFile is called by doSaveCommand, doSaveUsingFSSPec, and doSaveUsingFSRef. In conjunction with two supporting functions, it writes the document to disk using the "safe-save" procedure.

The first two lines retrieve a handle to the document structure and the file system specification from the document structure.

The next two lines create a temporary file name which is bound to be unique. FindFolder finds the temporary folder on the file's volume, or creates a temporary folder if necessary. FSMakeFSSpec makes a file system specification structure for the temporary file, using the volume reference number and parent directory ID returned by the FindFolder call. FSpCreate creates the temporary file in that directory on that volume, and FSpOpenDF opens the file's data fork.

Within the next if block, the appropriate function is called to write the document's data to the temporary file.

The two calls to FSClose close both the temporary and existing files prior to the call to FSpExchangeFiles, which swaps the files' data. The temporary file is then deleted and the data fork of the existing file is re-opened.

The function doCopyResources is called to copy the missing application name string resource from the resource fork of the application file to the resource fork of the new document file. If the file type is 'PICT', a 'pnot' resource and associated 'PICT' resource is also copied to the resource fork.

doReadTextFile

doReadTextFile is called by doOpenFile and the event function askSaveDiscardEventFunction to read in data from an open file of type 'TEXT'.

The first two lines retrieve the file reference number from the document structure.

The next three lines retrieve the handle to the TextEdit structure from the document structure and modify the text size and line height fields of the TextEdit structure.

SetFPos sets the file mark to the beginning of the file. GetEOF gets the number of bytes in the file. If the number of bytes exceeds that which can be stored in a TextEdit structure (32,767), the number of bytes which will be read from the file is restricted to 32,767.

NewHandle allocates a buffer equal to the size of the file (or 32,767 bytes if the preceding if statement executed). FSRead reads the data from the file into the buffer. MoveHHi and HLockHi move the buffer high in the heap and lock it preparatory

to the call to TESetText. TESetText copies the text in the buffer into the existing hText handle of the TextEdit edit structure. The buffer is then unlocked and disposed of.

doReadPictFile

doReadPictFile is called by doOpenFile and the event function askSaveDiscardEventFunction to read in data from an open file of type 'PICT'.

The first two lines retrieve the file reference number from the document structure. GetEOF gets the number of bytes in the file. SetFPos sets the file mark 512 bytes (the size of a 'PICT' file's header) past the beginning of the file, and the next line subtracts the header size from the total size of the file. NewHandle allocates memory for the Picture structure and FSRead reads in the file's data.

doWriteTextData

doWriteTextData is called by doWriteFile to write text data to the specified file.

The first two lines retrieve the handle to the TextEdit structure from the document structure. The number of bytes of text is then retrieved from the teLength field of the TextEdit structure.

SetFPos sets the file mark to the beginning of the file. FSWrite writes the specified number of bytes to the file. SetEOF adjusts the file's size. FlushVol stores to disk all unwritten data currently in the volume buffer.

The penultimate line sets the windowTouched field of the document structure to indicate that the document data on disk equates to the document data in memory.

doWritePictData

doWritePictData is called by doWriteFile to write picture data to the specified file.

The first two lines retrieve the handle to the relevant Picture structure from the document structure. SetFPos sets the file mark to the start of the file. FSWrite writes zeros in the first 512 bytes (the size of a 'PICT' file's header). GetHandleSize gets the size of the Picture structure and FSWrite writes the bytes in the Picture structure to the file. SetEOF adjusts the file's size and FlushVol stores to disk all unwritten data currently in the volume buffer.

The penultimate line sets the windowTouched field of the document structure to indicate that the document data on disk equates to the document data in memory.

getFilePutFileEventFunction

getFilePutFileEventFunction is the event (callback) function pertaining to the Open, Save Location, and Choose a Folder dialogs. It responds to button clicks in those dialogs.

When the user has clicked one of the dialog's buttons, the kNavCBUserAction message is received in the callBackSelector formal parameter. When this message is received, the first action is to call NavDialogGetReply to get a NavReplyRecord structure containing information about the dialog session. NavDialogGetUserAction is then called to get the user action which dismissed the dialog.

If the user clicked the Open button in an Open dialog, AECountItems is called to count the number of descriptor structures in the descriptor list returned in the selection field of the NavReplyRecord structure, and which is created from FSSpec references to items selected in the Open dialog. The for loop repeats for each of the descriptor structures. AEGetNthPtr gets the file system specification into a local variable of type FSSpec. This file system specification is then passed in the first parameter of a call to FSpGetFInfo, allowing the file type to be ascertained. The file system specification and file type are then passed in a call to the function doOpenFile, which creates a new window and reads in the file.

If the user clicked the Save button in a Save Location dialog, the window reference received in the callBackUD formal parameter is assigned to the local variable windowRef. (Recall that the window reference for the front window was passed

in the fifth parameter of the call to `NavCreatePutFileDialog`.) The next task is to determine which of the two file saving functions (`doSaveUsingFSSpec` or `doSaveUsingFSRef`) should be called to save the file. Accordingly, `AECoerceDesc` is called in an attempt to coerce the descriptor structure in the selection field of the `NavReplyRecord` structure to type `FSRef`. If the call is successful, `doSaveUsingFSRef` is called; if not, `doSaveUsingFSSpec` is called.

If the user clicked the Choose button in a Choose a Folder dialog, `AEGetNthPtr` is called to get the file system specification into a local variable of type `FSSpec`. When a file system specification describes a directory, as it does in this case, the name field is empty and the parID field contains the directory ID of that directory, not the ID of the parent directory. In this demonstration, the volume reference number and directory ID are passed in a call to `FSMakeFSSpec`, which fills in the fields of the `FSSpec` record pointed to by the fourth parameter. The contents of the fields of this `FSSpec` structure (the directory name, its parent directory ID, and the volume reference number) are then drawn in the bottom of the front window.

Before exit from the `kNavCBUserAction` case, `NavDisposeReply` is called to release the memory allocated for the `NavReplyRecord` structure.

When the user has clicked a dialog's Cancel button, the `kNavCBTerminate` message is received in the callBackSelector formal parameter. When this message is received, if a dialog reference has been assigned to the global variable `gModalToApplicationNavDialogRef` (as it will be in the case of the application-modal Open and Choose a Folder dialogs), the dialog is disposed of and the global variable is assigned `NULL`. If `gModalToApplicationNavDialogRef` contains `NULL`, the window reference received in the callBackUD formal parameter is assigned to the local variable `windowRef`. (Recall that the window reference for the front window was passed in the fifth parameter of the call to `NavCreatePutFileDialog`.) A handle to the window's document structure is then retrieved, allowing access to the dialog reference stored in that structure. The dialog is disposed of and the relevant field of the document structure is assigned `NULL`.

Note that, in Carbon applications, there is no need to respond to the `kNavCBEvent` message in this event function or the following event function in order to call the application's window updating function. This is assuming the standard window event handler is installed on the relevant windows, the application registers for the `kEventWindowDrawContent` event type, and calls its window updating function when that event type is received. The following example is provided for those circumstances in which these conditions are not met:

```
case kNavCBEvent:
  switch(callBackParms->eventData.eventDataParms.event->what)
  {
    case updateEvt:
      windowRef = (WindowRef) callBackParms->eventData.eventDataParms.event->message;
      if(GetWindowKind(windowRef) != kDialogWindowKind)
        doUpdate((EventRecord *) callBackParms->eventData.eventDataParms.event);
      break;
  }
  break;
```

askSaveDiscardEventFunction

`askSaveDiscardEventFunction` is the event (callback) function pertaining to the Save Changes and Discard Changes alerts. It responds to button clicks in those dialogs.

When the user has clicked one of the dialog's buttons, the `kNavCBUserAction` message is received in the `callBackSelector` formal parameter. When this message is received, the first action is to get a handle to the front window's document structure. (Recall that the reference to the front window was passed in the third parameter of the `NavCreateAskSaveChangesDialog` and `NavCreateAskDiscardChangesDialog` calls.) The main if block executes only if the `modalToWindowNavDialogRef` field of the document structure contains a dialog reference.

If the user clicked the Save button in a Save Changes alert, `doSaveCommand` is called to save the file and execution falls through to the `kNavUserActionDontSaveChanges` case where `doCloseDocWindow` is called to close the file, flush the volume, and close down the window.

If the user clicked the Don't Save button in a Save Changes alert, and if the program is running on Mac OS X,

doCloseDocWindow is called to close the file, flush the volume, and close down the window. If the program is running on Mac OS 9, the global variable gCloseDocWindow is set to true, causing the doCloseDocWindow call to occur in the function doCreateAskSaveChangesDialog. Before all this occurs, NavDialogDispose is called to dispose of the alert before the window is closed by the call to doCloseDocWindow.

If the user clicked the OK button in a Discard Changes alert, the window's content area is erased and the appropriate function (doReadTextFile or doReadPictFile) is called depending on whether the file type is 'TEXT' or 'PICT'. In addition, the window's "touched" field in the document structure is set to false and InvalWindowRect is called to force a redraw of the window's content region. Just before the InvalWindowRect call, SetWindowModified is called with false passed in the modified parameter. This causes the window proxy icon to appear in the enabled state, indicating no unsaved changes. The Discard Changes alert is then disposed of.

If the user clicked the Cancel button in a Save Changes or Discard Changes alert, and if it is a Save Changes alert, the flag gQuittingApplication is set to false. This has the effect of defeating the execution of the last block of code in the function doCloseDocWindow. (Recall that the isAskSaveChangesDialog field of the window's document structure is set to true when such alerts are created.) The alert is then disposed of.

doCopyResources

doCopyResources is called by doWriteFile. It copies the missing application name string resource from the resource fork of the application file to the resource fork of the new file. If the file type is PICT, a 'pnot' resource and associated 'PICT' resource is also copied. If the program is running on Mac OS 8/9 and the file type is TEXT, a 'pnot' resource, together with a 'TEXT' resource created within the function, are also copied. (For 'TEXT' files, previews are automatically created on Mac OS X.)

The first line retrieves a handle to the file's document structure. The next four lines establish the file type involved. FSpCreateResFile creates the resource fork in the new file and FSpOpenResFile opens the resource fork. The function for copying specified resources between specified files (doCopyAResource) is then called to copy the missing application name string resource from the resource fork of the application file to the resource fork of the new file.

If the file type is PICT, a 'pnot' resource and associated PICT resource is copied so as to provide a preview for PICT files in the Open dialog. (Of course, in a real application, the 'pnot' and PICT resource would be created by the application for each separate PICT file.)

If the program is running on Mac OS 8/9 and the file type is TEXT, a 'pnot' resource is copied and a TEXT resource is created and copied so as to provide a a preview for 'TEXT' files in the Open dialog. After the 'pnot' resource is copied, a relocatable block is created and 1024 bytes of the text in the TextEdit structure is copied to that block. AddResource turns that arbitrary data in memory into a TEXT resource, assigns a resource type, ID, and name to that resource, and inserts an entry in the resource map for the current resource file (in this case, the resource fork of the TEXT file). UpdateResFile then writes the resource map and data to disk.

CloseResFile closes the resource fork of the new file.

doCopyAResource

doCopyAResource copies specified resources between specified files. In this program, it is called only by doCopyResources.

UseResFile sets the application's resource fork as the current resource file. GetResource reads the specified resource into memory.

GetResInfo, given a handle, gets the resource type, ID and name. (Note that this line is included only because of the generic nature of doCopyResource. The calling function has passed doCopyResource the type and ID in this instance.)

DetachResource removes the resource's handle from the resource map without removing the resource from memory, and converts the resource handle into a generic handle. UseResFile makes the new file's resource fork the current resource file. AddResource makes the now arbitrary data in memory into a resource, assigns a resource ID, type and name to that resource, and inserts an entry in the resource map for the current resource file. UpdateResFile then writes the resource

map and data to disk.

SynchroniseFiles.c

doSynchroniseFiles

`doSynchroniseFiles` is called from `doIdle` when the installed timer fires (every 15 ticks when a document window is the front window).

A reference to the front non-floating window is obtained. The while loop walks the document window section of the window list (see the call to `GetNextWindow` at the bottom of the loop) looking for associated files whose locations have changed. When the last window in the list has been examined, the loop exits.

Within the while loop, `GetWRefCon` is called to retrieve the handle to the window's document structure.

If the `aliasHdl` field of the window's document structure contains `NULL`, the window does not yet have a file associated with it, in which case execution falls through to the next iteration of the while loop and the next window is examined.

If the window has an associated file, the handle to the associated alias structure, which contains the alias data for the file, is retrieved. `ResolveAlias` is then called to perform a search for the target of the alias, returning the file system specification for the target file in the third parameter. After identifying the target, `ResolveAlias` compares some key information about the target with the information in the alias structure. If the information differs, `ResolveAlias` updates the alias structure to match the target and sets the `aliasChanged` parameter to true.

If the `aliasChanged` parameter is set to true, meaning that the location of the file has changed, the `fileFSSpec` field of the window's document structure is assigned the file system specification structure returned by `ResolveAlias`. Since it is also possible that the user has renamed the file, `SetWTitle` is called to set the window's title to the filename contained in the name field of the file system specification structure returned by `ResolveAlias`.

The next task is to determine whether the user has moved the file to the trash or to a folder in the trash, in which case the document must be closed.

`FindFolder` is called to get the volume reference number and parent directory ID of the trash folder. (Note that `kUserDomain` is passed in the `vRefNum` parameter. On Mac OS 8/9, this is mapped to `kOnSystemDisk`.)

The do/while loop walks up the parent folder hierarchy to the root folder. At the first line in the do/while loop, if the root folder has been reached (`fsRtParID` is the parent ID of the root directory), the file is not in the trash, in which case the loop exits at that point. At the next if statement, the volume reference number and parent directory ID of the file are compared with the volume reference number and directory ID of the trash. If they match, the file is closed, its associated memory is disposed of, and the window is disposed of.

The bottom line of the do/while loop effects the walk up the parent directory hierarchy, `FSMakeFSSpec` creates a file system specification structure from the current contents of the `vRefNum` and `parID` fields of `newFSSPec`. Since `newFSSpec` is also the target, the parID field is "filled in" again, at every iteration of the loop, with the parent ID of the directory passed in the second parameter of the `FSMakeFSSpec` call.

ChooseAFolderDialog.c

doChooseAFolderDialog

`doChooseAFolderDialog`, which is called when the user chooses the Choose a Folder Dialog item in the demonstration menu, creates and displays a Choose a Folder dialog.

`NavGetDefaultDialogCreationOptions` initialises the specified NavDialogCreation Options structure with the defaults. `GetIndString` retrieves a Pascal string, which is converted to a `CFString` and assigned to the message field of a `NavDialogOptions` structure. This will appear immediately below the browser list in the Mac OS 8/9 dialog and above the browser list in the Mac OS X dialog.

The next line ensures that the dialog will be application-modal.

`NavCreateChooseFolderDialog` creates the dialog and `NavDialogRun` displays it.

Chapter 19. More on Resources and Core Foundation Preferences

IN THIS CHAPTER

- Introduction
- Search Path for Resources
- Releasing and Detaching Resources
- Creating, Opening, and Closing Resource Forks
- Reading and Manipulating Resources
- Writing Resources
- Partial Resources
- Application Preferences
- Main Resource Manager Constants, Data Types and Functions
- Main Core Foundation Preferences Services Functions
- Demonstration Program MoreResources Listing
- Demonstration Program MoreResources Comments

Introduction

Chapter 1 covered the basics of creating standard resources for an application's resource file and with reading in standard resources from application files and the system. In addition, the demonstration programs in preceding chapters have all involved the reading in of standard resources from those files.

This chapter is concerned with aspects of resources not covered in Chapter 1, including search paths, detaching and copying resources, creating, opening, and closing resource files, and reading from and writing to resource files. In addition, the accompanying demonstration program demonstrates the creation of **custom resources,** together with reading such resources from, and writing them to, the resource forks of files other than application and System files.

This chapter also addresses the matter of storing and retrieving application preferences using Core Foundation Preferences Services.

Search Path for Resources

Preamble

The Resource Manager follows a defined search path to find a resource when you use a Resource Manager function to read, or perform an operation on, a resource,. The different files whose resource forks may constitute the search path are therefore of some relevance. The following summarizes the typical locations of resources used by an application:

Resource Fork of	***Typical Resources Therein***	***Comments***
System file	Sounds (Mac OS 8/9), icons, cursors, and so on.	On startup, the Resource Manager creates a special zone within the system heap, creates a resource map which points to system-resident resources, opens the resource fork of the System file, and reads its resource map into memory.
Application file	Descriptions of menus, windows, controls, icons, and so on. Text used in dialogs, help balloons, and so on.	When a user opens an application, system software automatically opens the application's resource fork.
Application's Preferences file	The user's global preferences for the application.	An application should typically open the Preferences file at application launch, and leave it open.
Application's document file	Information specific only to the document, such as its window's last size and location.	When an application opens a document file, it should typically open the file's resource fork as well as its data fork.

Current Resource File

When your application opens the resource fork of a file, that file becomes the **current**

resource file. However, your application can change the current resource file, if necessary, using the function `UseResFile`.

The current resource file is always the first file in the search path. It is also the file on which most Resource Manager functions assume they should operate.

> **Note**
>
> The resource fork of a file is also called the resource file because, in some respects, you can treat it as if it were a separate file.

Default Search Order

During its search for a resource, if the Resource Manager cannot find the resource in the current resource file, it continues searching until it either finds the resource or has searched all files in the search path.

The Resource Manager normally looks first in the resource map in memory of the last resource fork your application opened. If the resource is not found there, the resource maps of each resource file open to your application are searched in reverse order of opening. If the resource is not found there, your application's resource map is searched. If the resource has still not been found, the searche continues in the System file's resource map.

Implications of the Default Search Order

The implications of this search order are that it allows your application to:

- Access resources in the System file.
- Override resources in the System file.
- Override your application's resources with document-specific resources.
- Share a resource amongst multiple files by storing it in the application's resource fork.

Setting the Current Resource File To Dictate the Search Order

You can, of course, rely on the Resource Manager's search order to find a particular resource; however, it is best to set the current resource file to the file containing the desired resource before reading and writing resource data. This ensures that that file will be searched first, thus possibly obviating unnecessary searches of other files.

As previously stated, the function `UseResFile` is used to set the current resource file. Note that `UseResFile` takes as its single parameter a **file reference number,** which

identifies an access path to the resource fork. The Resource Manager assigns a resource file a file reference number when it opens that file.

You can get the file reference number of the current resource file using the function `CurResFile`.

Restricting the Search to the Current Resource File

The search path may be restricted to the current resource file by using Resource Manager functions (such as `Get1Resource`), which look only in the current resource file's resource map when searching for a specific resource.

Releasing and Detaching Resources

When you have finished using a resource, you typically call `ReleaseResource`, which sets the handle's master pointer to `NULL` and releases the memory associated with the resource.

Your application can use `DetachResource` to replace a resource's handle in the resource map with `NULL` without releasing the associated memory. You will then be able to access the resource's data directly, without the aid of Resource Manager functions, and you will be able to pass the handle to a function which does not accept a resource handle. For example, the `AddResource` function, which makes arbitrary data in memory into a resource, requires a handle to data, not a handle to a resource.

`DetachResource` is useful when you want to copy a resource. The procedure is to read in the resource using `GetResource`, detach the resource to disassociate it from its resource file, and then copy the resource to a destination file using `AddResource`.

Creating, Opening, and Closing Resource Forks

Opening an Application's Resource Fork

As previously stated, the system software automatically opens your application's resource fork at application launch. The only action required by your application at launch is to call `CurResFile` to save the file reference number for the application's resource fork.

Creating and Opening a Resource Fork

Creating a Resource Fork

If your application needs to save resources to the resource fork of a file, and assuming the resource fork does not already exist, it must first create the resource fork. A call to `FSpCreateResFile` will create the resource fork. `FSpCreateResFile` requires four parameters: a file system specification structure, the application's signature, the file type, the script code. The effect of `FSpCreateResFile` varies as follows:

- If the file specified by the file system specification structure does not already exist, the function
    - Creates a file with an empty resource fork and resource map.
    - Sets the creator, type, and script code fields of the file's catalog information structure to the specified values.
- If the data fork of the file specified by the file system specification structure already exists but the file has a zero-length resource fork, the function
    - Creates an empty resource fork and resource map.
    - Changes the creator, type, and script code fields of the catalog information structure of the file to the specified values.
- If the file specified by the file system specification structure already exists and includes a resource fork with a resource map, the function does nothing.

Opening a Resource Fork

After creating a resource fork, and before attempting to write to it, you must open it using

`FSpOpenResFile`. `FSpOpenResFile` returns a file reference number which, as previously stated, may be used to change or limit the Resource Manager's search order.

> **Note**
>
> Note that, although the file reference number for the data fork and the resource fork usually match, you should not assume that this is always the case.

As previously stated, when you open a resource fork, the Resource Manager resets the search path so that the file whose resource fork you just opened becomes the current resource file.

After opening a resource fork, you can use Resource Manager functions to write resources to it.

> **Note**
>
> It is possible to write to the resource fork using File Manager functions. However, in general, you should always use Resource Manager functions.

Closing a Resource Fork

When you are finished using a resource fork that your application explicitly opened, you should close it using `CloseResFile`. Note that the Resource Manager automatically closes any resource forks opened by your application that are still open when your application calls `ExitToShell`.

Reading and Manipulating Resources

Depending on which Resource Manager function is used to read resources from a resource fork, you specify the resource to be read by either its resource type and resource ID or its resource type and resource name.

Reading From the Resource Map Without Loading the Resource

Those Resource Manager functions that return handles to resources normally read the resource data into memory if it is not already there. Sometimes, however, you may want to read, say, resource types and attributes from the resource map without reading the resource data into memory. Calling `SetResLoad` with the `load` parameter set to `false` causes subsequent calls to those functions which return handles to resources to *not* load the resource data to memory. (To read the resource data into memory after a call to `SetResLoad` with the `load` parameter set to `false`, call `LoadResource`.)

If you call `SetResLoad` with the `load` parameter set to `false`, be sure to call it again with the parameter set to `true` as soon as possible. Other parts of the system software that call the Resource Manager rely on the default setting (that is, the `load` parameter set to `true`), and some functions will not work properly if resources are not loaded automatically.

Indexing Through Resources

The Resource Manager provides functions that let you index through all resources of a given type (for example, using `CountResources` and `GetIndResource`). This can be useful when you want to read all resources of a given type.

Writing Resources

When you have opened a resource fork, you can write resources to it (assuming it is the current resource file).

A call to `AddResource` creates a new entry for a resource in the resource map in memory (but not on the disk) and sets that entry's location to refer to that resource's data. A call to either `UpdateResFile` or `WriteResFile` will then write the resource to disk. Note that you usually need to set the current resource file to the desired file before calling `AddResource` because `AddResource` always adds the resource to the resource map in memory, which corresponds to the current resource file.

When you change a resource referenced through the resource map in memory, you should call `ChangedResource` to set the `resChanged` attribute of that resource's resource map entry. `ChangedResource` reserves enough disk space to contain the changed resource. Immediately after calling `ChangedResource`, you should call `UpdateResFile` or `WriteResFile` to write the changed resource data to disk.

The difference between `UpdateResFile` and `WriteResFile` is as follows:

- `UpdateResFile` writes to disk only those resources that have been added or changed. It also writes the entire resource map to disk, overwriting its previous contents.
- `WriteResFile` writes only a single resource to disk and does not update the resource's entry in the resource map on disk.

Care with Purgeable Resources

If you are changing purgeable resources, you should use the Memory Manager function `HNoPurge` to ensure that the Resource Manager does not purge the resource while your application is in the process of changing it.

Partial Resources

Some resources, such as `'snd '` (Mac OS 8/9) and `'sfnt'` resources, can be too large to fit into available memory. In these cases, you can read a portion of the resource into memory, or alter a section of the resource while it is still on disk, using the functions `ReadPartialResource` and `WritePartialResource`.

Application Preferences

Many applications allow the user to set application preferences, which the application stores in a preferences file and retrieves when the application is launched. On Mac OS 8/9, your Preferences file should be located in the special folder titled Preferences provided by the operating system. The Preferences folder is located in the System folder. On Mac OS X, preferences are stored in the Preferences folder in the Library folder in each user's home directory (~/Library/Preferences).

You can save your application's preferences as a custom resource to the resource fork of your preferences file. Alternatively, and more correctly in the Mac OS X era, you can use the methodology provided by Core Foundation Preferences Services.

Using Core Foundation Preferences Services

Using Core Foundation Preferences Services involves storing values associated with a key, the key being used to later retrieve the value. The concept is similar to that applying to the key/value pairs used in information property lists (see Chapter 9).

Application ID

The name of the file in which Preference Services stores preferences information is specified by an **application ID.** For Mac OS X, this should be the same as the name associated with the `CFBundleIdentifier` key in the information property list in your application's `'plst'` resource. It thus takes the form of a Java package name, that is, a unique domain name followed by the application's name (for example, `com.MyCompany.MyApplication`). For Mac OS 8/9, it is sufficient to simply use the application's name. (Indeed, this abbreviation will be essential if the application ID in Java package form exceeds the 31-character limit for Mac OS 8/9 file names.)

Setting, Storing, and Retrieving a Preference

You use the function `CFPreferencesSetAppValue` to set a preferences value. The following example sets a value, which specifies that the application should use a full screen display:

```
CFStringRef applicationID = CFSTR("com.MyCompany.MyApplication");
CFStringRef fullScreenKey = CFSTR("fullScreen");
CFStringRef yes           = CFSTR("yes");
CFStringRef no            = CFSTR("no");

CFPreferencesSetAppValue(fullScreenKey,yes,applicationID);
```

`CFPreferencesSetAppValue` stores the value in a cache owned by your application. To flush the cache to permananent storage, you must call the function `CFPreferencesAppSynchronize`, passing in the application ID.

You would use the function `CFPreferencesGetAppBooleanValue` to retrieve the value, as in the following example:

```
CFStringRef applicationID = CFSTR("com.MyCompany.MyApplication");
CFStringRef fullScreenKey = CFSTR("fullScreen");
Boolean     booleanValue, success;

booleanValue = CFPreferencesGetAppBooleanValue(fullScreenKey,applicationID,&success);
```

You can use `CFPreferencesGetAppIntegerValue` to retrieve integer values (which are stored as strings) in the same way.

Preference Domains

Preference Services uses the concept of **domains** when creating or searching for a preference. User name, host name, and application ID identify a domain. `CFPreferencesSetAppValue` uses the current user and "any host" domain qualifiers

as the default, which is why only the application ID is passed in the function call. Alternative low-level Preferences Services functions are available to specify an exact domain.

Main Resource Manager Constants, Data Types and Functions

Constants

Resource Attributes

```
resSysHeap   = 64  System or application heap?
resPurgeable = 32  Purgeable resource?
resLocked    = 16  Load it in locked?
resProtected = 8   Protected?
resPreload   = 4   Load in on OpenResFile?
resChanged   = 2   Resource changed?
```

Data Types

```
typedef unsigned long  FourCharCode;
typedef FourCharCode   ResType;
```

Functions

Initializing the Resource Manager

```
short   InitResources(void);
```

Checking for Errors

```
short   ResError(void);
```

Creating an Empty Resource Fork

```
void    FSpCreateResFile(const FSSpec *spec,OSType creator,OSType fileType,ScriptCode
        scriptTag);
```

Opening Resource Forks

```
short   FSpOpenResFile(const FSSpec *spec,SignedByte permission);
```

Getting and Setting the Current Resource File

```
void    UseResFile(short refNum);
short   CurResFile(void);
short   HomeResFile(Handle theResource);
```

Reading Resources into Memory

```
Handle  GetResource(ResType theType,short theID);
Handle  Get1Resource(ResType theType,short theID);
Handle  GetNamedResource(ResType theType,ConstStr255Param name);
Handle  Get1NamedResource(ResType theType,ConstStr255Param name);
void    SetResLoad(Boolean load);
void    LoadResource(Handle theResource);
```

Getting and Setting Resource Information

```
void    GetResInfo(Handle theResource,short *theID,ResType *theType,Str255 name);
void    SetResInfo(Handle theResource,short theID,ConstStr255Param name);
```

```
short   GetResAttrs(Handle theResource);
void    SetResAttrs(Handle theResource,short attrs);
```

Modifying Resources

```
void    ChangedResource(Handle theResource);
void    AddResource(Handle theResource,ResType theType,short theID,ConstStr255Param name);
```

Writing to Resource Forks

```
void    UpdateResFile(short refNum);
void    WriteResource(Handle theResource);
```

Getting a Unique Resource ID

```
short   UniqueID(ResType theType);
short   Unique1ID(ResType theType);
```

Counting and Listing Resource Types

```
short   CountResources(ResType theType);
short   Count1Resources(ResType theType);
Handle  GetIndResource(ResType theType,short index);
Handle  Get1IndResource(ResType theType,short index);
short   CountTypes(void);
short   Count1Types(void);
void    GetIndType(ResType *theType,short index);
void    Get1IndType(ResType *theType,short index);
```

Getting Resource Sizes

```
long    GetResourceSizeOnDisk(Handle theResource);
long    GetMaxResourceSize(Handle theResource);
```

Disposing of Resources and Closing Resource Forks

```
void    ReleaseResource(Handle theResource);
void    DetachResource(Handle theResource);
void    RemoveResource(Handle theResource);
void    CloseResFile(short refNum);
```

Getting and Setting Resource Fork Attributes

```
short   GetResFileAttrs(short refNum);
void    SetResFileAttrs(short refNum,short attrs);
```

Main Core Foundation Preferences Services Functions

```
void    CFPreferencesSetAppValue(CFStringRef key,CFPropertyListRef value,
        CFStringRef applicationID);
void    CFPreferencesSetValue(CFStringRef key,CFPropertyListRef value,
        CFStringRef applicationID,CFStringRef userName,CFStringRef hostName);
Boolean CFPreferencesAppSynchronize(CFStringRef applicationID);
Boolean CFPreferencesSynchronize(CFStringRef applicationID,CFStringRef userName,
        CFStringRef hostName);
Boolean CFPreferencesGetAppBooleanValue(CFStringRef key,CFStringRef applicationID,
        Boolean *keyExistsAndHasValidFormat);
CFIndex CFPreferencesGetAppIntegerValue(CFStringRef key,CFStringRef applicationID,
        Boolean *keyExistsAndHasValidFormat);
```

Demonstration Program MoreResources Listing

```
//
//****************************************************************************************
// MoreResources.c                                                    CARBON EVENT
MODEL
//
//****************************************************************************************
//
// This program uses custom resources to:
//
// •  Store application preferences in the resource fork of a Preferences file.
//
// •  Store, in the resource fork of a document file:
//
//    •  The size and position of the window associated with the document.
//
//    •  A flattened PMPageFormat object containing information about how the pages of the
//       document should be printed, for example, on what paper size, in what printable
//       area, and in what orientation (landscape or portrait).
//
// The program also demonstrates setting, storing, and retrieving application preferences
// using Core Foundation Preferences Services.
//
// The program utilises the following standard resources:
//
// •  A 'plst' resource.
//
// •  An 'MBAR' resource, and 'MENU' resources for OS9Apple/Applcation, File, Edit and
//    Demonstration menus (preload, non-purgeable).
//
// •  A 'DLOG' resource (purgeable) and associated 'dlgx', 'DITL' and 'CNTL' resources
//    (purgeable) associated with the display of, and user modification of, current
//    application preferences.
//
// •  A 'STR#' resource (purgeable) containing the required name of the preferences file
//    created by the program.
//
// •  A 'STR ' resource (purgeable) containing the application-missing string, which is
copied
//    to the resource fork of the preferences file.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// The program creates and utilises the following custom resources:
//
// •  A 'PrFn' (preferences) resource comprising three boolean values, which is located in
the
//    program's resource file, which contains default preference values, and which is copied
//    to the resource fork of a preferences file created when the program is run for the
first
//    time. Thereafter, the 'PrFn' resource in the preferences file is used for the storage
//    and retrieval of application preferences set by the user.
//
// •  A 'WiSp' (window size and position) resource, which is created in the resource fork of
//    the document file used by the program, and which is used to store the associated
//    window's port rectangle converted to global coordinates.
//
// •  A 'PgFt' (page format) resource, which is created in the resource fork of the document
//    file used by the program, and which is used to store a flattened PMPageFormat object.
//
```

```
//
**********************************************************************************************

// ..........................................................................
includes

#include <Carbon.h>

// ..........................................................................
defines

#define rMenubar          128
#define mAppleApplication 128
#define  iAbout           1
#define mFile             129
#define  iOpen            2
#define  iClose           4
#define  iPageSetup       9
#define  iQuit            12
#define mEdit             130
#define  iPreferences     10
#define rPrefsDialog      128
#define  iSound           4
#define  iFullScreen      5
#define  iAutoScroll      6
#define rStringList       128
#define  iPrefsFileName   1
#define rTypePrefs        'PrFn'
#define  kPrefsID         128
#define rTypeWinSizePos   'WiSp'
#define  kWinSizePosID    128
#define rPageFormat       'PgFt'
#define  kPageFormatID    128
#define rTypeAppMiss      'STR '
#define  kAppMissID       -16397
#define topLeft(r)        (((Point *) &(r))[0])
#define botRight(r)       (((Point *) &(r))[1])

// ..........................................................................
typedefs

typedef struct
{
  Boolean sound;
  Boolean fullScreen;
  Boolean autoScroll;
  SInt16  programRunCount;
}  appPrefs, **appPrefsHandle;
typedef struct
{
  Rect windowRect;
}  windowSizePos, **windowSizePosHandle;

typedef struct
{
  Handle pageFormatHdl;
  FSSpec fileFSSpec;
}  docStructure, **docStructureHandle;

// ..................................................................... global
variables

Boolean        gRunningOnX = false;
PMPrintSession gPrintSession;
WindowRef      gWindowRef;
SInt16         gAppResFileRefNum;
```

```
Boolean        gSoundPref;
Boolean        gFullScreenPref;
Boolean        gAutoScrollPref;
SInt16         gProgramRunCountPref;
Boolean        gSoundCFPref;
Boolean        gFullScreenCFPref;
Boolean        gAutoScrollCFPref;
SInt16         gProgramRunCountCFPref;
Boolean        gWindowOpen        = false;
Boolean        gPageFormatChanged = false;
SInt16         gPrefsFileRefNum   = 0;

// ..................................................................... function
prototypes

void      main                              (void);
void      doPreliminaries                   (void);
OSStatus  appEventHandler                   (EventHandlerCallRef,EventRef,void *);
OSStatus  windowEventHandler                (EventHandlerCallRef,EventRef,void *);
void      doDrawContent                     (WindowRef);
void      doAdjustMenus                     (void);
void      doMenuChoice                      (MenuID,MenuItemIndex);
void      doErrorAlert                      (SInt16);
void      doOpenCommand                     (void);
void      navEventFunction                  (NavEventCallbackMessage,NavCBRecPtr,
                                            NavCallBackUserData);
void      doOpenWindow                      (FSSpec);
void      doCloseWindow                     (void);
void      doPreferencesDialog               (void);
OSStatus  doPageSetupDialog                 (void);
void      doGetPreferencesResource          (void);
void      doGetPreferencesCFPrefs           (void);
OSErr     doCopyResource                    (ResType,SInt16,SInt16,SInt16);
void      doSavePreferencesResource         (void);
void      doSavePreferencesCFPrefs          (void);
void      doLoadAndSetWindowSizeAndPosition (WindowRef);
void      doGetFrameWidthAndTitleBarHeight  (WindowRef,SInt16 *,SInt16 *);
void      doSaveWindowSizeAndPosition       (WindowRef);
void      doGetPageFormat                   (WindowRef);
void      doSavePageFormat                  (WindowRef);
// ****************************************************************************
main

void  main(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  OSStatus      osStatus;
  EventTypeSpec applicationEvents[] = { { kEventClassCommand, kEventProcessCommand  } ,
                                        { kEventClassMenu,    kEventMenuEnableItems }  } ;

  // ..................................................................... do
preliminaries

  doPreliminaries();

  // ................................ set current resource file to application resource
fork

  gAppResFileRefNum = CurResFile();

  // ..................................................... set up menu bar and
menus

  menubarHdl = GetNewMBar(rMenubar);
```

```
  if(menubarHdl == NULL)
    doErrorAlert(MemError());
  SetMenuBar(menubarHdl);
  DrawMenuBar();

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
    }
    menuRef = GetMenuRef(mEdit);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iPreferences);
      DeleteMenuItem(menuRef,iPreferences - 1);
      DisableMenuItem(menuRef,0);
    }

    EnableMenuCommand(NULL,kHICommandPreferences);

    gRunningOnX = true;
  }
  else
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
      SetMenuItemCommandID(menuRef,iQuit,kHICommandQuit);
    menuRef = GetMenuRef(mEdit);
    if(menuRef != NULL)
      SetMenuItemCommandID(menuRef,iPreferences,kHICommandPreferences);
  }

  // ..................................................................... create printing
session

  osStatus = PMCreateSession(&gPrintSession);
  if(osStatus != kPMNoError)
    doErrorAlert(osStatus);

  // ............................ read in application preferences and increment program
run count

  doGetPreferencesResource();
  doGetPreferencesCFPrefs();

  gProgramRunCountPref++;
  gProgramRunCountCFPref++;

  // ..................................................... install application event
handler

  InstallApplicationEventHandler(NewEventHandlerUPP((EventHandlerProcPtr) appEventHandler),
                                 GetEventTypeCount(applicationEvents),applicationEvents,
                                 0,NULL);

  // ..................................................... run application event
loop

  RunApplicationEventLoop();
}

// ********************************************************************************
```

```
doPreliminaries

void  doPreliminaries(void)
{
  MoreMasterPointers(320);
  InitCursor();
}

// ************************************************************************
appEventHandler

OSStatus  appEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                          void * userData)
{
  OSStatus      result = eventNotHandledErr;
  UInt32        eventClass;
  UInt32        eventKind;
  HICommand     hiCommand;
  MenuID        menuID;
  MenuItemIndex menuItem;

  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);
  switch(eventClass)
  {
    case kEventClassCommand:
      if(eventKind == kEventProcessCommand)
      {
        GetEventParameter(eventRef,kEventParamDirectObject,typeHICommand,NULL,
                          sizeof(HICommand),NULL,&hiCommand);
        menuID = GetMenuID(hiCommand.menu.menuRef);
        menuItem = hiCommand.menu.menuItemIndex;
        if(hiCommand.commandID == kHICommandPreferences)
        {
          doPreferencesDialog();
          result = noErr;
        }
        else if(hiCommand.commandID == kHICommandQuit)
        {
          while(FrontWindow())
            doCloseWindow();
          PMRelease(&gPrintSession);
          doSavePreferencesResource();
          doSavePreferencesCFPrefs();
        }
        else if((menuID >= mAppleApplication && menuID <= mEdit))
        {
          doMenuChoice(menuID,menuItem);
          result = noErr;
        }
      }
      break;

    case kEventClassMenu:
      if(eventKind == kEventMenuEnableItems)
      {
        doAdjustMenus();
        result = noErr;
      }
      break;
  }

  return result;
}

// ************************************************************************
```

windowEventHandler

```
OSStatus  windowEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                             void* userData)
{
  OSStatus  result = eventNotHandledErr;
  UInt32    eventClass;
  UInt32    eventKind;
  WindowRef windowRef;
  Rect      mainScreenRect;
  BitMap    screenBits;
  Point     idealHeightAndWidth, minimumHeightAndWidth;
  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);

  switch(eventClass)
  {
    case kEventClassWindow:
      GetEventParameter(eventRef,kEventParamDirectObject,typeWindowRef,NULL,sizeof
(windowRef),
                        NULL,&windowRef);
      switch(eventKind)
      {
        case kEventWindowDrawContent:
          doDrawContent(windowRef);
          result = noErr;
          break;

        case kEventWindowGetIdealSize:
          mainScreenRect = GetQDGlobalsScreenBits(&screenBits)->bounds;
          idealHeightAndWidth.v = mainScreenRect.bottom - 75;
          idealHeightAndWidth.h = 600;
          SetEventParameter(eventRef,kEventParamDimensions,typeQDPoint,
                            sizeof(idealHeightAndWidth),&idealHeightAndWidth);
           result = noErr;
          break;

        case kEventWindowGetMinimumSize:
          minimumHeightAndWidth.v = 190;
          minimumHeightAndWidth.h = 400;
          SetEventParameter(eventRef,kEventParamDimensions,typeQDPoint,
                            sizeof(minimumHeightAndWidth),&minimumHeightAndWidth);
          result = noErr;
          break;

        case kEventWindowClose:
          doCloseWindow();
          result = noErr;
          break;
      }
      break;
  }

  return result;
}

// ********************************************************************************
doDrawContent

void  doDrawContent(WindowRef windowRef)
{
  RGBColor           whiteColor  = {  0xFFFF, 0xFFFF, 0xFFFF } ;
  RGBColor           blueColor  = {  0x1818, 0x4B4B, 0x8181 } ;
  Rect               portRect;
  docStructureHandle docStrucHdl;
  PMPageFormat       pageFormat  = kPMNoPageFormat;
```

```
Str255             string;
PMResolution       resolution;
PMRect             paperRect;
PMRect             pageRect;
UInt16             orientation;
RGBForeColor(&whiteColor);
RGBBackColor(&blueColor);
GetWindowPortBounds(windowRef,&portRect);
EraseRect(&portRect);

SetPortWindowPort(windowRef);

MoveTo(10,20);
TextFace(bold);
DrawString("\pApplication Preferences:");
MoveTo(10,35);
DrawString("\pResource Fork Prefs File");
MoveTo(170,35);
DrawString("\pCF Preferences Services");
TextFace(normal);
MoveTo(10,50);
DrawString("\pSound On:  ");
if(gSoundPref)  DrawString("\pYES");
else  DrawString("\pNO");
MoveTo(10,65);
DrawString("\pFull Screen On:  ");
if(gFullScreenPref)  DrawString("\pYES");
else  DrawString("\pNO");
MoveTo(10,80);
DrawString("\pAutoScroll On:  ");
if(gAutoScrollPref)  DrawString("\pYES");
else  DrawString("\pNO");
MoveTo(10,95);
DrawString("\pProgram run count: ");
NumToString((SInt32) gProgramRunCountPref,string);
DrawString(string);

MoveTo(170,50);
DrawString("\pSound On:  ");
if(gSoundCFPref)  DrawString("\pYES");
else  DrawString("\pNO");
MoveTo(170,65);
DrawString("\pFull Screen On:  ");
if(gFullScreenCFPref)  DrawString("\pYES");
else  DrawString("\pNO");
MoveTo(170,80);
DrawString("\pAutoScroll On:  ");
if(gAutoScrollCFPref)  DrawString("\pYES");
else  DrawString("\pNO");
MoveTo(170,95);
DrawString("\pProgram run count: ");
NumToString((SInt32) gProgramRunCountCFPref,string);
DrawString(string);

docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
PMUnflattenPageFormat((*docStrucHdl)->pageFormatHdl,&pageFormat);
PMGetResolution(pageFormat,&resolution);
PMGetAdjustedPaperRect(pageFormat,&paperRect);
PMGetAdjustedPageRect(pageFormat,&pageRect);
PMGetOrientation(pageFormat,&orientation);
if(pageFormat != kPMNoPageFormat)
  PMRelease(&pageFormat);

MoveTo(10,115);
TextFace(bold);
DrawString("\pInformation From Document's 'PgFt' (Page Format) Resource:");
```

```
  TextFace(normal);

  if((*docStrucHdl)->pageFormatHdl != NULL)
  {
    MoveTo(10,130);
    DrawString("\pApplication's Drawing Resolution:  ");
    NumToString((long) resolution.hRes,string);
    DrawString(string);
    DrawString("\p dpi horizontal, ");
    NumToString((long) resolution.vRes,string);
    DrawString(string);
    DrawString("\p dpi vertical");

    MoveTo(10,145);
    DrawString("\pPaper Rectangle Size in Drawing Resolution:  ");
    NumToString((long) (paperRect.bottom - paperRect.top),string);
    DrawString(string);
    DrawString("\p by ");
    NumToString((long) (paperRect.right - paperRect.left),string);
    DrawString(string);

    MoveTo(10,160);
    DrawString("\pPage Rectangle Size in Drawing Resolution:  ");
    NumToString((long) (pageRect.bottom - pageRect.top),string);
    DrawString(string);
    DrawString("\p by ");
    NumToString((long) (pageRect.right - pageRect.left),string);
    DrawString(string);

    MoveTo(10,175);
    DrawString("\pOrientation:  ");
    if(orientation == 1)
      DrawString("\pPortrait");
    else if(orientation == 2)
      DrawString("\pLandscape");
  }
  else
  {
    MoveTo(10,130);
    DrawString("\pA page format ('PgFt') resource has not been saved yet.");
    MoveTo(10,145);
    DrawString("\pOpen the Page Setup... dialog before closing the window or quitting.");
  }

  QDFlushPortBuffer(GetWindowPort(gWindowRef),NULL);
}
// ********************************************************************************
doAdjustMenus

void  doAdjustMenus(void)
{
  MenuRef menuRef;

  if(gWindowOpen)
  {
    menuRef = GetMenuRef(mFile);
    DisableMenuItem(menuRef,iOpen);
    EnableMenuItem(menuRef,iClose);
    EnableMenuItem(menuRef,iPageSetup);
  }
  else
  {
    menuRef = GetMenuRef(mFile);
    EnableMenuItem(menuRef,iOpen);
    DisableMenuItem(menuRef,iClose);
    DisableMenuItem(menuRef,iPageSetup);
```

```
  }

  DrawMenuBar();
}

// **********************************************************************************
doMenuChoice

void  doMenuChoice(MenuID menuID,MenuItemIndex menuItem)
{
  OSStatus osStatus;
  Rect     portRect;

  if(menuID == 0)
    return;

  switch(menuID)
  {
    case mAppleApplication:
      if(menuItem == iAbout)
        SysBeep(10);
      break;

    case mFile:
      switch(menuItem)
      {
        case iClose:
          doCloseWindow();
          break;

        case iOpen:
          doOpenCommand();
          break;
        case iPageSetup:
          osStatus = doPageSetupDialog();
          if(osStatus != kPMNoError && osStatus != kPMCancel)
            doErrorAlert(osStatus);
          if(FrontWindow())
          {
            GetWindowPortBounds(FrontWindow(),&portRect);
            InvalWindowRect(FrontWindow(),&portRect);
          }
          break;
      }
      break;
  }
}

// **********************************************************************************
doErrorAlert

void  doErrorAlert(SInt16 errorCode)
{
  Str255 errorString;
  SInt16 itemHit;

  NumToString((SInt32) errorCode,errorString);

  if(errorCode != memFullErr)
    StandardAlert(kAlertCautionAlert,errorString,NULL,NULL,&itemHit);
  else
  {
    StandardAlert(kAlertStopAlert,errorString,NULL,NULL,&itemHit);
    ExitToShell();
  }
}
```

```
// ****************************************************************************
doOpenCommand

void  doOpenCommand(void)
{
  OSErr            osError = noErr;
  NavDialogOptions dialogOptions;
  NavEventUPP      navEventFunctionUPP;
  NavReplyRecord   navReplyStruc;
  SInt32           index, count;
  AEKeyword        theKeyword;
  DescType         actualType;
  FSSpec           fileSpec;
  Size             actualSize;

  osError = NavGetDefaultDialogOptions(&dialogOptions);

  if(osError == noErr)
  {
    navEventFunctionUPP = NewNavEventUPP((NavEventProcPtr) navEventFunction);
    osError = NavGetFile(NULL,&navReplyStruc,&dialogOptions,navEventFunctionUPP,NULL,NULL,
                         NULL,NULL);
    DisposeNavEventUPP(navEventFunctionUPP);
    if(osError == noErr && navReplyStruc.validRecord)
    {
      osError = AECountItems(&(navReplyStruc.selection),&count);
      if(osError == noErr)
      {
        for(index=1;index<=count;index++)
        {
          osError = AEGetNthPtr(&(navReplyStruc.selection),index,typeFSS,&theKeyword,
                                &actualType,&fileSpec,sizeof(fileSpec),&actualSize);

          doOpenWindow(fileSpec);
        }
      }

      NavDisposeReply(&navReplyStruc);
    }
  }
}

// ****************************************************************************
navEventFunction

void  navEventFunction(NavEventCallbackMessage callBackSelector,NavCBRecPtr callBackParms,
                       NavCallBackUserData callBackUD)
{
}

// ****************************************************************************
doOpenWindow

void  doOpenWindow(FSSpec fileSpec)
{
  OSStatus           osError;
  docStructureHandle docStrucHdl;
  Rect               contentRect = {  100,100,290,500 } ;
  WindowAttributes   attributes  = {  kWindowStandardHandlerAttribute |
                                      kWindowStandardDocumentAttributes } ;
  EventTypeSpec      windowEvents[] = {  {  kEventClassWindow,
kEventWindowDrawContent    } ,
                                         {  kEventClassWindow, kEventWindowGetIdealSize   } ,
                                         {  kEventClassWindow, kEventWindowGetMinimumSize } ,
                                         {  kEventClassWindow,
```

```
kEventWindowClose          }  } ;

  osError = CreateNewWindow(kDocumentWindowClass,attributes,&contentRect,&gWindowRef);

  if(osError != noErr)
    QuitApplicationEventLoop();

  if(!(docStrucHdl = (docStructureHandle) NewHandle(sizeof(docStructure))))
  {
    DisposeWindow(gWindowRef);
    QuitApplicationEventLoop();
  }

  SetWRefCon(gWindowRef,(SInt32) docStrucHdl);
  (*docStrucHdl)->fileFSSpec = fileSpec;
  SetWTitle(gWindowRef,(*docStrucHdl)->fileFSSpec.name);
  (*docStrucHdl)->pageFormatHdl = NULL;
  SetPortWindowPort(gWindowRef);
  UseThemeFont(kThemeSmallSystemFont,smSystemScript);

  InstallWindowEventHandler(gWindowRef,
                            NewEventHandlerUPP((EventHandlerProcPtr) windowEventHandler),
                            GetEventTypeCount(windowEvents),windowEvents,0,NULL);

  doLoadAndSetWindowSizeAndPosition(gWindowRef);
  doGetPageFormat(gWindowRef);
  ShowWindow(gWindowRef);
  gWindowOpen = true;
}

// ************************************************************************
doCloseWindow

void  doCloseWindow(void)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  OSErr              osError = 0;

  windowRef = FrontWindow();
  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

  doSaveWindowSizeAndPosition(windowRef);

  if(gPageFormatChanged)
    doSavePageFormat(windowRef);

  DisposeHandle((Handle) docStrucHdl);
  DisposeWindow(windowRef);
  gWindowOpen = false;
}

// ************************************************************************
doPreferencesDialog

void  doPreferencesDialog(void)
{
  DialogRef  modalDlgRef;
  ControlRef controlHdl;
  SInt16     itemHit;
  Rect       portRect;

  if(!(modalDlgRef = GetNewDialog(rPrefsDialog,NULL,(WindowRef) -1)))
    return;

  SetDialogDefaultItem(modalDlgRef,kStdOkItemIndex);
```

```
  SetDialogCancelItem(modalDlgRef,kStdCancelItemIndex);

  GetDialogItemAsControl(modalDlgRef,iSound,&controlHdl);
  SetControlValue(controlHdl,gSoundPref);
  GetDialogItemAsControl(modalDlgRef,iFullScreen,&controlHdl);
  SetControlValue(controlHdl,gFullScreenPref);
  GetDialogItemAsControl(modalDlgRef,iAutoScroll,&controlHdl);
  SetControlValue(controlHdl,gAutoScrollPref);
  ShowWindow(GetDialogWindow(modalDlgRef));

  do
  {
    ModalDialog(NULL,&itemHit);
    GetDialogItemAsControl(modalDlgRef,itemHit,&controlHdl);
    SetControlValue(controlHdl,!GetControlValue(controlHdl));
  }  while((itemHit != kStdOkItemIndex) && (itemHit != kStdCancelItemIndex));

  if(itemHit == kStdOkItemIndex)
  {
    GetDialogItemAsControl(modalDlgRef,iSound,&controlHdl);
    gSoundPref = gSoundCFPref = GetControlValue(controlHdl);
    GetDialogItemAsControl(modalDlgRef,iFullScreen,&controlHdl);
    gFullScreenPref = gFullScreenCFPref = GetControlValue(controlHdl);
    GetDialogItemAsControl(modalDlgRef,iAutoScroll,&controlHdl);
    gAutoScrollPref = gAutoScrollCFPref = GetControlValue(controlHdl);
  }

  DisposeDialog(modalDlgRef);

  if(gWindowRef)
  {
    GetWindowPortBounds(gWindowRef,&portRect);
    InvalWindowRect(gWindowRef,&portRect);
  }

  doSavePreferencesResource();
  doSavePreferencesCFPrefs();
}

// ************************************************************************
doPageSetupDialog

OSStatus  doPageSetupDialog(void)
{
  docStructureHandle docStrucHdl;
  OSStatus           osStatus    = kPMNoError;
  PMPageFormat       pageFormat  = kPMNoPageFormat;
  Boolean            userClickedOKButton;

  docStrucHdl = (docStructureHandle) GetWRefCon(gWindowRef);
  HLock((Handle) docStrucHdl);

  if((*docStrucHdl)->pageFormatHdl == NULL)
  {
    osStatus = PMCreatePageFormat(&pageFormat);
    if((osStatus == kPMNoError) && (pageFormat != kPMNoPageFormat))
      osStatus = PMSessionDefaultPageFormat(gPrintSession,pageFormat);
    if(osStatus == kPMNoError)
      osStatus = PMFlattenPageFormat(pageFormat,&(*docStrucHdl)->pageFormatHdl);
  }
  else
  {
    osStatus = PMUnflattenPageFormat((*docStrucHdl)->pageFormatHdl,&pageFormat);
    if(osStatus == kPMNoError)
      osStatus = PMSessionValidatePageFormat(gPrintSession,pageFormat,kPMDontWantBoolean);
  }
```

```
  if((osStatus == kPMNoError) && (pageFormat != kPMNoPageFormat))
  {
    SetThemeCursor(kThemeArrowCursor);

    osStatus = PMSessionPageSetupDialog(gPrintSession,pageFormat,&userClickedOKButton);
    if(!userClickedOKButton)
      osStatus = kPMCancel;
  }

  if(osStatus == kPMNoError && userClickedOKButton)
  {
    DisposeHandle((*docStrucHdl)->pageFormatHdl);
    (*docStrucHdl)->pageFormatHdl = NULL;
    osStatus = PMFlattenPageFormat(pageFormat,&(*docStrucHdl)->pageFormatHdl);

    gPageFormatChanged = true;
  }

  if(pageFormat != kPMNoPageFormat)
    PMRelease(&pageFormat);

  HUnlock((Handle) docStrucHdl);

  return osStatus;
}

// *********************************************************************
doGetPreferencesResource

void  doGetPreferencesResource(void)
{
  Str255         prefsFileName;
  OSErr          osError;
  SInt16         volRefNum;
  long           directoryID;
  FSSpec         fileSSpec;
  SInt16         fileRefNum;
  appPrefsHandle appPrefsHdl;

  GetIndString(prefsFileName,rStringList,iPrefsFileName);

  osError = FindFolder(kUserDomain,kPreferencesFolderType,kDontCreateFolder,&volRefNum,
                       &directoryID);

  if(osError == noErr)
    osError = FSMakeFSSpec(volRefNum,directoryID,prefsFileName,&fileSSpec);
  if(osError == noErr || osError == fnfErr)
    fileRefNum = FSpOpenResFile(&fileSSpec,fsCurPerm);

  if(fileRefNum == -1)
  {
    FSpCreateResFile(&fileSSpec,'PpPp','pref',smSystemScript);
    osError = ResError();

    if(osError == noErr)
    {
      fileRefNum = FSpOpenResFile(&fileSSpec,fsCurPerm);
      if(fileRefNum != -1 )
      {
        UseResFile(gAppResFileRefNum);
        osError = doCopyResource(rTypePrefs,kPrefsID,gAppResFileRefNum,fileRefNum);
        if(osError == noErr)
          osError = doCopyResource(rTypeAppMiss,kAppMissID,gAppResFileRefNum,fileRefNum);
        if(osError != noErr)
        {
          CloseResFile(fileRefNum);
```

```
          osError = FSpDelete(&fileSSpec);
          fileRefNum = -1;
        }
      }
    }
  }

  if(fileRefNum != -1)
  {
    UseResFile(fileRefNum);

    appPrefsHdl = (appPrefsHandle) Get1Resource(rTypePrefs,kPrefsID);
    if(appPrefsHdl == NULL)
      return;

    gSoundPref = (*appPrefsHdl)->sound;
    gFullScreenPref = (*appPrefsHdl)->fullScreen;
    gAutoScrollPref = (*appPrefsHdl)->autoScroll;
    gProgramRunCountPref = (*appPrefsHdl)->programRunCount;

    gPrefsFileRefNum = fileRefNum;

    UseResFile(gAppResFileRefNum);
  }
}

// **********************************************************************
doGetPreferencesCFPrefs

void  doGetPreferencesCFPrefs(void)
{
  CFStringRef applicationID9 = CFSTR("MoreResources CFPrefs");
  CFStringRef applicationIDX = CFSTR("com.Windmill.MoreResources");
  CFStringRef applicationID;
  CFStringRef soundOnKey     = CFSTR("sound");
  CFStringRef fullScreenKey  = CFSTR("fullScreen");
  CFStringRef autoScrollKey  = CFSTR("autoScroll");
  CFStringRef runCountKey    = CFSTR("runCount");
  Boolean     booleanValue, success;

  if(gRunningOnX)
    applicationID = applicationIDX;
  else
    applicationID = applicationID9;

  booleanValue = CFPreferencesGetAppBooleanValue(soundOnKey,applicationID,&success);
  if(success)
    gSoundCFPref = booleanValue;
  else
  {
    gSoundCFPref      = true;
    gFullScreenCFPref = true;
    gAutoScrollCFPref = true;
    doSavePreferencesCFPrefs();
    return;
  }

  booleanValue = CFPreferencesGetAppBooleanValue(fullScreenKey,applicationID,&success);
  if(success)
    gFullScreenCFPref = booleanValue;

  booleanValue = CFPreferencesGetAppBooleanValue(autoScrollKey,applicationID,&success);
  if(success)
    gAutoScrollCFPref = booleanValue;

  gProgramRunCountCFPref = CFPreferencesGetAppIntegerValue(runCountKey,applicationID,
```

```
                                                        &success);
}

// ****************************************************************************
doCopyResource

OSErr  doCopyResource(ResType resType,SInt16 resID,SInt16 sourceFileRefNum,
                      SInt16 destFileRefNum)
{
  SInt16  oldResFileRefNum;
  Handle  sourceResourceHdl;
  ResType ignoredType;
  SInt16  ignoredID;
  Str255  resourceName;
  SInt16  resAttributes;
  OSErr   osError;

  oldResFileRefNum = CurResFile();
  UseResFile(sourceFileRefNum);

  sourceResourceHdl = Get1Resource(resType,resID);

  if(sourceResourceHdl != NULL)
  {
    GetResInfo(sourceResourceHdl,&ignoredID,&ignoredType,resourceName);
    resAttributes = GetResAttrs(sourceResourceHdl);
    DetachResource(sourceResourceHdl);
    UseResFile(destFileRefNum);
    if(ResError() == noErr)
      AddResource(sourceResourceHdl,resType,resID,resourceName);
    if(ResError() == noErr)
      SetResAttrs(sourceResourceHdl,resAttributes);
    if(ResError() == noErr)
      ChangedResource(sourceResourceHdl);
    if(ResError() == noErr)
      WriteResource(sourceResourceHdl);
  }
  osError = ResError();

  ReleaseResource(sourceResourceHdl);
  UseResFile(oldResFileRefNum);

  return osError;
}

// ****************************************************************
doSavePreferencesResource

void  doSavePreferencesResource(void)
{
  SInt16         currentResFile;
  appPrefsHandle appPrefsHdl;
  Handle         existingResHdl;
  Str255         resourceName = "\pPreferences";

  if(gPrefsFileRefNum == -1)
    return;

  currentResFile = CurResFile();

  appPrefsHdl = (appPrefsHandle) NewHandleClear(sizeof(appPrefs));

  HLock((Handle) appPrefsHdl);

  (*appPrefsHdl)->sound = gSoundPref;
  (*appPrefsHdl)->fullScreen = gFullScreenPref;
```

```
  (*appPrefsHdl)->autoScroll = gAutoScrollPref;
  (*appPrefsHdl)->programRunCount = gProgramRunCountPref;

  UseResFile(gPrefsFileRefNum);

  existingResHdl = Get1Resource(rTypePrefs,kPrefsID);

  if(existingResHdl != NULL)
  {
    RemoveResource(existingResHdl);
    DisposeHandle(existingResHdl);
    if(ResError() == noErr)
      AddResource((Handle) appPrefsHdl,rTypePrefs,kPrefsID,resourceName);
    if(ResError() == noErr)
      WriteResource((Handle) appPrefsHdl);
  }

  HUnlock((Handle) appPrefsHdl);

  ReleaseResource((Handle) appPrefsHdl);
  UseResFile(currentResFile);
}
// **************************************************************
doSavePreferencesCFPrefs

void  doSavePreferencesCFPrefs(void)
{
  CFStringRef applicationID9 = CFSTR("MoreResources CFPrefs");
  CFStringRef applicationIDX = CFSTR("com.Windmill.MoreResources");
  CFStringRef applicationID;
  CFStringRef soundOnKey    = CFSTR("sound");
  CFStringRef fullScreenKey = CFSTR("fullScreen");
  CFStringRef autoScrollKey = CFSTR("autoScroll");
  CFStringRef yes           = CFSTR("yes");
  CFStringRef no            = CFSTR("no");
  CFStringRef runCountKey   = CFSTR("runCount");
  Str255      runCountPascalString;
  CFStringRef runCountCFString;

  if(gRunningOnX)
    applicationID = applicationIDX;
  else
    applicationID = applicationID9;

  if(gSoundCFPref)
    CFPreferencesSetAppValue(soundOnKey,yes,applicationID);
  else
    CFPreferencesSetAppValue(soundOnKey,no,applicationID);

  if(gFullScreenCFPref)
    CFPreferencesSetAppValue(fullScreenKey,yes,applicationID);
  else
    CFPreferencesSetAppValue(fullScreenKey,no,applicationID);

  if(gAutoScrollCFPref)
    CFPreferencesSetAppValue(autoScrollKey,yes,applicationID);
  else
    CFPreferencesSetAppValue(autoScrollKey,no,applicationID);

  NumToString((SInt32) gProgramRunCountCFPref,runCountPascalString);
  runCountCFString = CFStringCreateWithPascalString(NULL,runCountPascalString,
                                                    CFStringGetSystemEncoding());
  CFPreferencesSetAppValue(runCountKey,runCountCFString,applicationID);

  CFPreferencesAppSynchronize(applicationID);
```

```
  if(runCountCFString != NULL)
    CFRelease(runCountCFString);
}
// *****************************************************
doLoadAndSetWindowSizeAndPosition

void  doLoadAndSetWindowSizeAndPosition(WindowRef windowRef)
{
  docStructureHandle  docStrucHdl;
  SInt16              fileRefNum;
  OSErr               osError;
  windowSizePosHandle windowSizePosHdl;
  Rect                windowRect;
  SInt16              frameWidth, titleBarHeight, menuBarHeight;

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  fileRefNum = FSpOpenResFile(&(*docStrucHdl)->fileFSSpec,fsRdWrPerm);
  if(fileRefNum < 0)
  {
    osError = ResError();
    doErrorAlert(osError);
    return;
  }

  windowSizePosHdl = (windowSizePosHandle) Get1Resource(rTypeWinSizePos,kWinSizePosID);

  if(windowSizePosHdl != NULL )
  {
    windowRect = (*windowSizePosHdl)->windowRect;
  }
  else
  {
    doGetFrameWidthAndTitleBarHeight(windowRef,&frameWidth,&titleBarHeight);
    GetThemeMenuBarHeight(&menuBarHeight);
    SetRect(&windowRect,frameWidth + 1,titleBarHeight + menuBarHeight + 1,
            frameWidth + 400,titleBarHeight + menuBarHeight + 190);
  }

  MoveWindow(windowRef,windowRect.left,windowRect.top,false);
  SizeWindow(windowRef,windowRect.right - windowRect.left,windowRect.bottom - windowRect.
top,
             true);

  ReleaseResource((Handle) windowSizePosHdl);
  CloseResFile(fileRefNum);
}

// *****************************************************
doGetFrameWidthAndTitleBarHeight

void  doGetFrameWidthAndTitleBarHeight(WindowRef windowRef,SInt16 *frameWidth,
                                       SInt16 *titleBarHeight)
{
  RgnHandle structureRegionHdl = NewRgn();
  RgnHandle contentRegionHdl   = NewRgn();
  Rect      structureRect, contentRect;

  GetWindowRegion(windowRef,kWindowStructureRgn,structureRegionHdl);
  GetRegionBounds(structureRegionHdl,&structureRect);
  GetWindowRegion(windowRef,kWindowContentRgn,contentRegionHdl);
  GetRegionBounds(contentRegionHdl,&contentRect);
  *frameWidth = contentRect.left - structureRect.left - 1;
  *titleBarHeight = contentRect.top - structureRect.top - 1;

  DisposeRgn(structureRegionHdl);
  DisposeRgn(contentRegionHdl);
```

```
}

// ************************************************************
doSaveWindowSizeAndPosition

void  doSaveWindowSizeAndPosition(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  Rect               portRect;
  SInt16             fileRefNum;
  OSErr               osError;
  windowSizePos       windowSizePosStruct;
  windowSizePosHandle windowSizePosHdl;

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  fileRefNum = FSpOpenResFile(&(*docStrucHdl)->fileFSSpec,fsRdWrPerm);
  if(fileRefNum < 0)
  {
    osError = ResError();
    doErrorAlert(osError);
    return;
  }

  GetWindowPortBounds(windowRef,&portRect);
  SetPortWindowPort(windowRef);
  LocalToGlobal(&topLeft(portRect));
  LocalToGlobal(&botRight(portRect));
  windowSizePosStruct.windowRect = portRect;

  windowSizePosHdl = (windowSizePosHandle) Get1Resource(rTypeWinSizePos,kWinSizePosID);
  if(windowSizePosHdl != NULL)
  {
    **windowSizePosHdl = windowSizePosStruct;
    ChangedResource((Handle) windowSizePosHdl);
    osError = ResError();
    if(osError != noErr)
      doErrorAlert(osError);
  }
  else
  {
    windowSizePosHdl = (windowSizePosHandle) NewHandle(sizeof(windowSizePos));
    if(windowSizePosHdl != NULL)
    {
      **windowSizePosHdl = windowSizePosStruct;
      AddResource((Handle) windowSizePosHdl,rTypeWinSizePos,kWinSizePosID,
                  "\pLast window size and position");
    }
  }
  if(windowSizePosHdl != NULL)
  {
    UpdateResFile(fileRefNum);
    osError = ResError();
    if(osError != noErr)
      doErrorAlert(osError);

    ReleaseResource((Handle) windowSizePosHdl);
  }

  CloseResFile(fileRefNum);
}

// ************************************************************************
doGetPageFormat

void  doGetPageFormat(WindowRef windowRef)
{
```

```
  docStructureHandle docStrucHdl;
  SInt16             fileRefNum;
  OSErr              osError;
  Handle             pageFormatResourceHdl = NULL;
  PMPageFormat       pageFormat  = kPMNoPageFormat;
  Boolean            settingsChanged;

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  HLock((Handle) docStrucHdl);

  fileRefNum = FSpOpenResFile(&(*docStrucHdl)->fileFSSpec,fsRdWrPerm);
  if(fileRefNum < 0)
  {
    osError = ResError();
    doErrorAlert(osError);
    return;
  }

  pageFormatResourceHdl = Get1Resource(rPageFormat,kPageFormatID);

  if(pageFormatResourceHdl != NULL)
  {
    PMUnflattenPageFormat(pageFormatResourceHdl,&pageFormat);
    PMSessionValidatePageFormat(gPrintSession,pageFormat,&settingsChanged);

    DisposeHandle((*docStrucHdl)->pageFormatHdl);
    PMFlattenPageFormat(pageFormat,&((*docStrucHdl)->pageFormatHdl));

    if(pageFormat != kPMNoPageFormat)
      PMRelease(&pageFormat);
    ReleaseResource(pageFormatResourceHdl);
  }

  CloseResFile(fileRefNum);

  HUnlock((Handle) docStrucHdl);
}
// ********************************************************************************
doSavePageFormat

void  doSavePageFormat(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  Handle             pageFormatHdl;
  SInt16             fileRefNum;
  OSErr              osError;
  Size               handleSize;

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  handleSize = GetHandleSize((*docStrucHdl)->pageFormatHdl);

  fileRefNum = FSpOpenResFile(&(*docStrucHdl)->fileFSSpec,fsRdWrPerm);
  if(fileRefNum < 0)
  {
    osError = ResError();
    doErrorAlert(osError);
    return;
  }

  pageFormatHdl = Get1Resource(rPageFormat,kPageFormatID);

  if(pageFormatHdl != NULL)
  {
    RemoveResource(pageFormatHdl);
    DisposeHandle(pageFormatHdl);
    pageFormatHdl = NewHandle(handleSize);
```

```
    BlockMove(*((*docStrucHdl)->pageFormatHdl),*pageFormatHdl,handleSize);
    AddResource(pageFormatHdl,rPageFormat,kPageFormatID,"\pPage Format");
    osError = ResError();
    if(osError != noErr)
      doErrorAlert(osError);
  }
  else
  {
    pageFormatHdl = NewHandle(handleSize);
    if(pageFormatHdl != NULL)
    {
      BlockMove(*((*docStrucHdl)->pageFormatHdl),*pageFormatHdl,handleSize);
      AddResource(pageFormatHdl,rPageFormat,kPageFormatID,"\pPage Format");
      osError = ResError();
      if(osError != noErr)
        doErrorAlert(osError);
    }
  }

  if(pageFormatHdl != NULL)
  {
    UpdateResFile(fileRefNum);
    osError = ResError();
    if(osError != noErr)
      doErrorAlert(osError);
    ReleaseResource(pageFormatHdl);
  }

  gPageFormatChanged = false;

  CloseResFile(fileRefNum);
}

//
*****************************************************************************************
```

Demonstration Program MoreResources Comments

This program uses two methods for saving and retrieving application preferences. The first method involves a custom preferences (`'PrFn'`) resource, which is saved to the resource fork of a preferences file named MoreResources Preferences. The second method involves the use of Core Foundation Preferences Services, which creates and saves preferences to a file named com.Windmill.MoreResources.plist on Mac OS X and MoreResources CFPrefs. plist on Mac OS 8/9. These files are created when the program is first run. Thereafter, the preferences will be read in from the preferences files every time the program is run and replaced whenever the user invokes the Preferences dialog to change the preferences settings.

When the application is first run, an "application missing" `'STR '` resource is also copied to the resource fork of the preferences file created by the first method. On Mac OS 8/9, if the user double-clicks on the icon for this preferences file, an alert is invoked displaying the text contained in the "application missing" resource.

After the program is launched, the user should choose Open from the File menu to open the included demonstration document file (titled "MoreResources Document"). The resource fork of this file contains two custom resources, namely, a `'WiSp'` resource containing the last saved window user state and zoom state, and a `'PgFt'` resource containing a flattened `PMPageFormat` object. These two resources are read in whenever the document file is opened. The `'WiSp'` resource is written to whenever the file is closed. The `'PgFt'` resource is written to when the file is closed only if the user invoked the Page Setup dialog while the document was open.

No data is read in from the document's data fork. Instead, the window is used to display the current preferences settings and information extracted from the document's `'PgFt'` resource. This is simply to keep the code not directly related to resources to a minimum.

The user should choose different paper size, scaling, and orientation settings in the Page Setup dialog, resize or zoom the window, close the file, reopen the file, and note that, the saved page format values are correctly retrieved and the window is reopened in the size and position in which is was closed. The user should also change the application preferences settings via the Preferences dialog (which is invoked when the Preference item in the Mac OS 8/9 Edit menu or Mac OS X Application menu is chosen), quit the program, relaunch the program, and note that the last saved preferences settings are retrieved at program launch.

The user may also care to remove the `'WiSp'` and `'PgFt'` resources from the document file's resource fork, run the program, force a `'PgFt'` resource to be created and written to by invoking the Page Setup dialog while the document file is open, quit the program, and

rerun the program, noting that `'WiSp'` and `'PgFt'` resources are created in the document file's resource fork if they do not already exist.

When done, the user may want to remove the preferences files from the Preferences folder.

defines

`rPrefsDialog` represents the `'DLOG'` resource ID for the Preferences dialog and the following three constants represent the item numbers of the dialog's checkboxes. `rStringList` and `iPrefsFileName` represent the `'STR#'` resource ID and index for the string containing the name of the application's preferences file created by the first method. The next eight constants represent resource types and IDs for the custom preferences resource, the custom window state resource, the custom page format resource, and the application missing string resource.

typedefs

The `appPrefs` data type is for the application preferences settings. The three Boolean fields are set by checkboxes in the Preferences dialog. The `SInt16` field is incremented each time the program is run.

The `windowSizePos` data type is for the window size and position. The `docStructure` data type is for the window's document structure.

Global Variables

`gAppResFileRefNum` will be assigned the file reference number for the application file's resource fork.

`gSoundPref`, `gFullScreenPref`, and `gAutoScrollPref` will hold the application preferences settings, as will `gSoundCFPref`, `gFullScreenCFPref`, and `gAutoScrollCFPref`. `gProgramRunCountPref` will hold the number of times the application has been run, as will `gProgramRunCountCFPref`.

`gPageFormatChanged` is set to true when the Page Setup dialog is invoked, and determines whether a new page format resource will be written to the document file when the file is closed.

`gPrefsFileRefNum` will be assigned the file reference number for the preferences file's resource fork.

main

The call to `CurResFile` sets the application's resource fork as the current resource file. If the application is running on Mac OS X, `DeleteMenuItem` is called to delete the Preferences item and its preceding divider from the Edit menu. (On Mac OS X, the Preferences command is in the Application menu.) Since, on Mac OS X, the Preferences item is disabled by default, `EnableMenuCommand` is called to enable that item.

The call to `PMCreateSession` creates a printing session.

The calls to `doGetPreferencesResource` and `doGetPreferencesCFPrefs` read in the application preferences settings from the preferences files.

- If the preferences file to which the `doGetPreferencesResource` call pertains does not exist, a preferences file will be created, default preferences settings will be copied to it from the application file, and these default settings will then be read in from the newly-created file.
- If the preferences file to which the `doGetPreferencesCFPrefs` call pertains does not exist, a a preferences file will be created by Core Foundation Preferences Services and default preferences settings will be copied to it.

appEventHandler

When the `kEventProcessCommand` event type is received, if the `commandID` field of the `HICommand` structure contains `'pref'`, the function `doPreferencesDialog` is called to display the Preferences dialog.

windowEventHandler

When the `kWindowEventClose` event type is received, the function `doCloseWindow` is called. Amongst other things, `doCloseWindow` calls the function `doSaveWindowSizeAndPosition`, which saves the windows position and size to the resource fork of the document file.

doDrawContent

`doDrawContent` simply prints the current preferences and page format information in the window for the information of the user.

doOpenCommand

`doOpenCommand` is a much simplified version of the actions normally taken when a user chooses the Open command from a File menu. Note that, in this program, `NavGetFile`, rather than the Navigation Services 3.0 functions `NavCreateGetFileDialog` and `NavDialogRun`, is used to create and display the Open dialog.

`NavGetFile` presents the Open dialog. If the user clicks the Open button, `doOpenWindow` is called.

doOpenWindow

`doOpenWindow` creates a new window, creates a block for a document structure, associates the document structure with the window object, assigns the file system specification for the chosen file to the document structure's file system specification field, sets the window's title, assigns `NULL` to the `pageFormatHdl` field of the document structure, and installs the window's event handler.

At that point, `doLoadAndSetWindowSizeAndPosition` is called to read in the window size and position (`'Wisp'`) resource from the document's resource fork, and to position and size the window accordingly. In addition, `doGetPageFormat` is called to read in the page format (`'PgFt'`) resource, and assign a handle to it to the `pageFormatHdl` field of the document structure.

With the window sized and positioned, `ShowWindow` is called to make the window visible (the window's 'WIND' resource specifies that the window is to be initially invisible) and a flag is set to indicate that the window is open.

navEventFunction

A universal procedure pointer to a navigation event (callback) function must be passed in the `eventProc` parameter of the `NavGetFile` call if the Open dialog is to be movable and resizable. However, in a Carbon event model application this function may be empty if its only purpose would be to call the window's updating function, and if the window receives, and responds to, the `kEventWindowDrawContent` event type.

doCloseWindow

`doCloseWindow` is called when the user chooses the Quit item, chooses Close from the File menu, or clicks the window's close box/button.

At the first two lines, a reference to the front window, and a handle to the associated document structure, are retrieved.

The call to `doSaveWindowSizeAndPosition` saves the window's port rectangle coordinates, converted to global coordinates, to the window size and position (`'WiSp'`) resource in the document's resource fork. If the Page Setup dialog was invoked while the window was open, and if the user dismissed the dialog by clicking the OK button, a call is made to `doSavePageFormat` to save the `PMPageFormat` object to the page format (`'PgFt'`) resource in the document file's resource fork.

`DisposeHandle` disposes of the document structure, `DisposeWindow` disposes of the window structure, and the last line sets the "window is open" flag to indicate that the window is not open.

doPreferencesDialog

`doPreferencesDialog` is called when the user chooses the Preferences item. The function presents the Preferences dialog and sets the values in the global variables, which hold the current application preferences according to the settings of the dialog's checkboxes.

Note that, at the last two lines, calls are made to `doSavePreferencesResource` and `doSavePreferencesCFPrefs`.

doPageSetupDialog

`doPageSetupDialog` is called when the user chooses the Page Setup item in the File menu. It presents the Page Setup dialog.

If the user dismisses the dialog with a click on the OK button, the flag, which indicates that a page format change has been made, is set to true.

doGetPreferencesResource

`doGetPreferencesResource`, which is called from the main function immediately after program launch, is the first of those functions central to the demonstration aspects of the program. Its purpose is to create the preferences file if it does not already exist, copying the default preferences resource and the missing application string resource to that file as part of the creation process, and to read in the preferences resource from the previously existing or newly created preferences file.

`GetIndString` retrieves from the application's resource fork the resource containing the required name of the preferences file ("MoreResources Preferences").

`FindFolder` finds the location of the Preferences folder, returning the volume reference number and directory ID in the last two parameters. (Note that `kUserDomain` is passed in the `vRefNum` parameter. On Mac OS 8/9, this is mapped to `kOnSystemDisk`.) `FSMakeFSSpec` makes a file system specification from the preferences file name, volume reference number and directory ID. This file system specification is used in the `FSpOpenResFile` call to open the resource fork of the preferences file with exclusive read/write permission.

If the specified file does not exist, `FSpOpenResFile` returns -1. In that event `FSpCreateResFile` creates the preferences file. The call to `FSpCreateResFile`

creates the file of the specified type on the specified volume in the specified directory and with the specified name and creator. (Note that the creator is set to an arbitrary signature, which no other application known to the Finder is likely to have. This is so that a double-click on the preferences file icon will cause the Finder to immediately display the missing application alert (Mac OS 8/9). Note also that, if `'pref'` is used as the `fileType` parameter, the icon used for the file will be the system-supplied preferences document icon, which looks like this:

If the file is created successfully, the resource fork of the file is opened by `FSpOpenResFile` and the master preferences (`'PrFn'`) and application missing `'STR '` resources are copied to the resource fork from the application's resource file. If the resources are not successfully copied, `CloseResFile` closes the resource fork of the new file, `FspDelete` deletes the file, and the `fileRefNum` variable is set to indicate that the file does not exist.

If the preferences file exists (either previously or newly created), `UseResFile` sets the resource fork of that file as the current resource file, the preferences resource is read in from the resource fork by `Get1Resource` and, if the read was successful, the three Boolean values and one `SInt16` value that constitute the application's preference settings are assigned to the global variables which store those values. (Note that, in this program, the function `Get1Resource` is used to read in resources so as to restrict the Resource Manager's search for the specified resource to the current resource file.)

The penultimate line assigns the file reference number for the open preferences file resource fork to a global variable. (The fork is left open). The last line resets the application's resource fork as the current resource file.

doGetPreferencesCFPrefs

`doGetPreferencesCFPrefs` is called from main to retrieve the application's preferences using Core Foundation Preferences Services routines.

Since the application ID in Java package name format (with ".plst" appended) exceeds the 31-character limit for Mac OS 8/9 filenames, an abbreviated ID is used if the program is running on Mac OS 8/9.

If the first call to `CFPreferencesGetAppBooleanValue` is successful, the preferences file exists and the Boolean value retrieved using the "sound" key is assigned to the relevant global variable. If this call is not successful, default values are assigned to the three global variables that hold the Boolean preferences values, and the function returns.

Two more calls to `CFPreferencesGetAppBooleanValue` with the appropriate keys retrieve the remaining two Boolean values and assign them to the relevant global variables. The call to `CFPreferencesGetAppIntegerValue` with the "run count" key retrieves the integer value representing the number of times the program has been run and

assigns it to the relevant global variable.

doCopyResource

`doCopyResource` is called by `doGetPreferences` to copy the default preferences and application missing string to the newly-created preferences file from the application file.

The first two lines save the current resource file's file reference number and set the application's resource fork as the current resource file. This will be the "source" file.

The `Get1Resource` call reads the specified resource into memory. `GetResInfo` gets the resource's name and `GetResAttrs` gets the resource's attributes. `DetachResource` replaces the resource's handle in the resource map with `NULL` without releasing the associated memory. The resource data is now simply arbitrary data in memory.

`UseResFile` sets the preferences file's resource fork as the current resource file. `AddResource` makes the arbitrary data in memory into a resource, assigning it the specified type, ID and name. `SetResAttrs` sets the resource attributes in the resource map. `ChangedResource` tags the resource for update and pre-allocates the required disk space. `WriteResource` then writes the resource to disk.

With the resource written to disk, `ReleaseResource` discards the resource in memory and `UseResFile` resets the resource file saved at the first line as the current resource file.

doSavePreferencesResource

`doSavePreferencesResource` is called when the user dismisses the Preferences dialog to save the new preference settings to the preferences file created by `doGetPreferencesResource`. It assumes that that preferences file is already open.

At the first two lines, if `doGetPreferences` was not successful in opening the preferences file at program launch, the function simply returns. The call to `CurResFile` saves the file reference number of the current resource file for later restoration.

The next six lines create a new preferences structure and assign to its fields the values in the global variables which store the current preference settings. `UseResFile` makes the preferences file's resource fork the current resource file. `Get1Resource` gets a handle to the existing preferences resource. Assuming the call is successful (that is, the preferences resource exists), `RemoveResource` is called to remove the resource from the resource map, `AddResource` is called to make the preferences structure in memory into a resource, and `WriteResource` is called to write the resource data to disk.

`ReleaseResource` disposes of the preferences structure in memory and UseResFile makes the previously saved resource file the current resource file.

doSavePreferencesCFPrefs

`doSavePreferencesCFPrefs` is called when the user dismisses the Preferences dialog to set and store the new preference settings using Core Foundation Preferences Services functions.

Since the application ID in Java package name format (with ".plst" appended) exceeds the 31-character limit for Mac OS 8/9 filenames, an abbreviated ID is used if the program is running on Mac OS 8/9.

For the three Boolean preferences values, `CFPreferencesSetAppValue` is called with the appropriate key to set the values in the application's preferences cache. For the program run count (integer) value, `NumToString` and `CFStringCreateWithPascalString` are called to convert the integer value to a `CFString` before `CFPreferencesSetAppValue` is called.

The call to `CFPreferencesAppSynchronize` flushes the cache to permanent storage, creating the preferences file if it does not already exist.

doLoadAndSetWindowSizeAndPosition

`doLoadAndSetWindowSizeAndPosition` gets the window size and position (`'WiSp'`) resource from the resource fork of the document file and moves and sizes the window according to retrieved size and position data.

`GetWRefCon` gets a handle to the window's document structure so that the file system specification can be retrieved and used in the `FSpOpenResFile` call to open the document file's resource fork.

`Get1Resource` attempts to read in the `'WiSp'` resource. If the `Get1Resource` call is successful, the retrieved data is assigned to a local variable of type `Rect`. (Recall that this is the window's port rectangle in global coordinates.) If `Get1Resource` call is not successful (as it will be if no `'WiSp'` resource currently exists in the document file's resource fork), a default size and position for the window is established.

The calls to `MoveWindow` and `SizeWindow` set the position and size of the window to either the retrieved or default position and size.

doSaveWindowSizeAndPosition

`doSaveWindowSizeAndPosition` saves the current window size and position (the window's port rectangle in global coordinates) to the document file's resource fork, and is called when the associated window is closed by the user or the Quit item is chosen.

The first line gets a handle to the window's document structure so that the document file's file system specification can be retrieved and used in the `FSpOpenResFile` call. If the resource fork cannot be opened, an error alert is presented and the function simply returns.

The next block gets the window's port rectangle, converts it to global coordinates, and assigns it to the `windowRect` field of a `windowSizePosStruct` structure.

`Get1Resource` attempts to read the `'WiSp'` resource from the document's resource fork into memory. If the `Get1Resource` call is successful, the resource in memory is made equal to the previously "filled-in" `windowSizePosStruct` structure and the resource is tagged as changed. If the `Get1Resource` call is not successful (that is, the document file's resource fork does not yet contain a `'WiSp'` resource), the else block creates a new `windowSizePosStruct` structure, makes this structure equal to the previously "filled-in" `windowSizePosStruct` structure, and makes this data in memory into a `'WiSp'` resource.

If an existing `'WiSp'` resource was successfully read in, or if a new `'WiSp'` resource was successfully created in memory, `UpdateResFile` writes the resource data and map to disk, and `ReleaseResource` discards the resource in memory. The document file's resource fork is then closed by `CloseResFile`.

doGetPageFormat

`doGetPageFormat` reads in the `'PgFt'` resource, which contains a flattened `PMPageFormat` object, from the document file's resource fork. The function is called when the document is opened.

The first line gets a handle to the window's document structure so that the document file's file system specification can be retrieved and used in the call to `FSpOpenResFile`. If the call is not successful, an error alert is presented and the function simply returns.

If the resource fork is successfully opened, the call to `Get1Resource` attempts to read in the resource. If the call is successful, the resource is unflattened into a `PMPageFormat` object, `PMValidatePageFormat` is called to ensure that the page format information is compatible with the driver for the current printer, and `PMFlattenPageFormat` is called to flatten the `PMPageFormat` object. When `PMFlattenPageFormat` returns, the second parameter (the relevant field of the window's document structure) contains a handle to the flattened data. The resource is then released and the document file's resource fork is closed.

doSavePageFormat

`doSavePageFormat` saves the flattened `PMPageFormat` object, whose handle is stored in the window's document structure, to a `'PgFt'` resource in the document file's resource fork. The function is called when the file is closed if the user invoked the PageSetup dialog while the document was open and dismissed the dialog by clicking the OK button.

The first line gets a handle to the window's document structure so that the document file's file system specification can be retrieved and used in the call to `FSpOpenResFile`. If the call is not successful, an error alert is presented and the function simply returns.

`Get1Resource` attempts to read the `'PgFt'` resource from the document's resource fork into memory. If the `Get1Resource` call is successful, the contents of the handle in the document structure's `pageFormatHdl` field are copied to the resource in memory, and the resource is tagged as changed. If the `Get1Resource` call is not successful (that is, the document file's resource fork does not yet contain a `'PgFt'` resource), a block of memory the size of a `'PgFt'` resource is allocated, the contents of the handle in the document structure's `pageFormatHdl` field are copied to that block, and `AddResource` makes that block into a `'PgFt'` resource.

If an existing `'PgFt'` resource was successfully read in, or if a new `'PgFt'` resource was successfully created in memory, `UpdateResFile` writes the resource map and data to disk. `ReleaseResource` then discards the resource in memory. At the last line, the document file's resource fork is then closed.

Chapter 20. Carbon Scrap

IN THIS CHAPTER

- The Carbon Scrap Manager and the Scrap
- Private Scrap
- TextEdit, Dialogs, and Scrap
- Main Carbon Scrap Manager Constants, Data Types, and Functions
- Demonstration Program CarbonScrap Listing
- Demonstration Program CarbonScrap Comments

The Carbon Scrap Manager and the Scrap

Introduction

The inclusion of the word "Carbon" in the title of this chapter is quite deliberate, reflecting the fact that, in Carbon, the original Scrap Manager has been redesigned to fully support the needs of the preemptively scheduled Mac OS X.

Applications which support cut, copy, and paste operations write data to, and read data from, the scrap. The scrap is a storage area, maintained by the **Scrap Manager,** which holds the last text, graphics, sounds, etc., cut or copied by the user.

The various data formats in which data may be written to, and read from, the scrap are called **scrap flavors.** A scrap flavor is a self-contained, self-describing stream of bytes that represent a discreet object such as a picture or text selection. Each scrap flavor has a **scrap flavor type** and a set of **scrap flavor flags.** The scrap may contain data in one or more flavors, each flavor being a different representation of the same object.

Your application specifies the scrap flavor, or flavors, to be read from, and written to, the scrap. The ultimate aim is to allow the user to copy and paste documents

- Within a document created by your application.
- Between different documents created by your application.
- Between documents created by your application and documents created by other applications.

Location of the Scrap

On Mac OS 8/9, space is allocated for the scrap in each application's heap. The system software stores a handle to the scrap of the current process in the system global variable `ScrapHandle`. When an application is launched, data is copied to the newly activated application's heap from the previously active application's heap. If the scrap is too large to fit in the application's heap, the scrap is copied to disk. In this event, the handle to the scrap is set to `NULL` to indicate that the scrap is on disk.

On Mac OS X, the scrap is held by the pasteboard server.

Scrap Reference

A scrap is referred to by a scrap reference. The data type `ScrapRef` is defined as a pointer to

a scrap reference:

```
typedef struct OpaqueScrapRef *ScrapRef;
```

Note that, although there is only one scrap, there may be multiple `ScrapRef` values. A `ScrapRef` value is valid only until the scrap is cleared.

Scrap Flavors

Standard Scrap Flavors

Your application should be capable of writing at least one of the following **standard scrap flavors** to the scrap and should be capable of reading both:

- `'TEXT'` (a series of ASCII characters in the same format as a `'TEXT'` resource).
- `'PICT'` (a QuickDraw picture in the same format as a `'PICT'` resource).

Optional Flavors

Your application may also choose to support the following optional scrap flavors:

- `'styl'` (a series of bytes that describe styled text data, and which have the same format as a TextEdit `'styl'` resource).
- `'movv'` (a series of bytes that define a movie, and which have the same format as a `'movv'` resource).

Private Flavors

It is also possible for your application to use its own private flavor, or flavors, but this should be in addition to at least one of the standard flavors.

Preferred Flavor

Recall that each flavor in the scrap (assuming there is more than one) is simply a different representation of the same object.

Your application should have a **preferred scrap flavor.** When reading data from the scrap, your application should request its preferred flavor first and only request its next preferred flavor if the preferred flavor does not exist in the scrap. When writing data to the scrap, your application should write its preferred flavor first. Any additional flavors should be written in the preferred order.

Implementing Edit Menu Commands

You use the Edit menu Cut, Copy, and Paste commands to implement cutting, copying, and pasting of data within or between documents. The following are the actions your application should perform to support these three commands:

Edit Command	*Actions Performed by Your Application*
Cut	If there is a current selection range, write the data in the selection range to the scrap and remove the data from the document.
Copy	If there is a current selection range, write the data in the selection range to the scrap.
Paste	Read the scrap and insert the data (if any) at the insertion point, replacing any current selection.

> **Note**
>
> The insertion point in a text document is represented by the blinking vertical bar known as the **caret.** There is a close relationship between the selection range and the insertion point in that the insertion point is, in effect, an empty selection range.

If your application implements a Clear command, it should remove the data in the current selection range but should not save the data to the scrap.

Cut and Copy-Putting Data in the Scrap

A typical approach to a basic implementation of the Cut and Copy commands is as follows:

- Determine whether the frontmost window is a document window or a dialog.
- If the frontmost window is a document window
    - Call `ClearCurrentScrap` to purge the current contents of the scrap.
    - Call `GetCurrentScrap` to obtain a reference to the current scrap.
    - Determine whether the current selection contains text or a picture.
    - If the current selection is text, get a pointer to the selected text and get the size of the selection. If the current selection is a picture, get a pointer to the picture

structure and get the size of that structure.

    - Call `PutScrapFlavor` to write the data to the scrap, passing the appropriate flavor type in the `flavorType` parameter.

    - If the command was the Cut command, delete the selection from the current document.

- If the frontmost window is a dialog, use the Dialog Manager functions `DialogCut` or `DialogCopy`, as appropriate, to write the selected data to the scrap.

Paste-Getting Data From the Scrap

When you read the data from the scrap, your application should request the data in the application's preferred flavor type. If your application determines that that flavor does not exist in the scrap, it should then request the data in another flavor. If your application does not have a preferred flavor type, it should read each flavor type that your application supports.

If you request a scrap format that is not in the scrap, the Scrap Manager uses the Translation Manager to convert any one of the scrap flavor types currently in the scrap into the scrap flavor requested by your application. The Translation Manager looks for a translator that can perform one of these translations. If such a translator is available, the Translation Manager uses the translator to translate the data in the scrap into the requested flavor.

A typical approach to an implementation of the Paste command, for an application that prefers a flavor type of `'TEXT'` as its first preference, is as follows:

- Determine whether the frontmost window is a document window or a dialog.

- If the frontmost window is a document window

    - Call `GetCurrentScrap` to obtain a reference to the current scrap.

    - Call `GetScrapFlavorFlags` to determine whether the preferred flavor exists in the scrap.

    - If the preferred flavor type (`'TEXT'`) does exist, call `GetScrapFlavorSize` to get the size of the text data, allocate a relocatable block of that size, and call `GetScrapFlavorData` to read the data into that block. Copy the data in the relocatable block to the current document at the insertion point.

    - If the preferred flavor type does not exist, call `GetScrapFlavorFlags` again to determine whether the next preferred flavor (say, `'PICT'`) exists in

the scrap. If it does, call `GetScrapFlavorSize` to get the size of the picture data, allocate a relocatable block of that size, and call `GetScrapFlavorData` to read the data into that block. Call `DrawPicture` to draw the picture described by the data in the relocatable block in the current document at the insertion point.

- If the frontmost window is a , use the Dialog Manager function `DialogPaste` to paste the text from the scrap in the dialog.

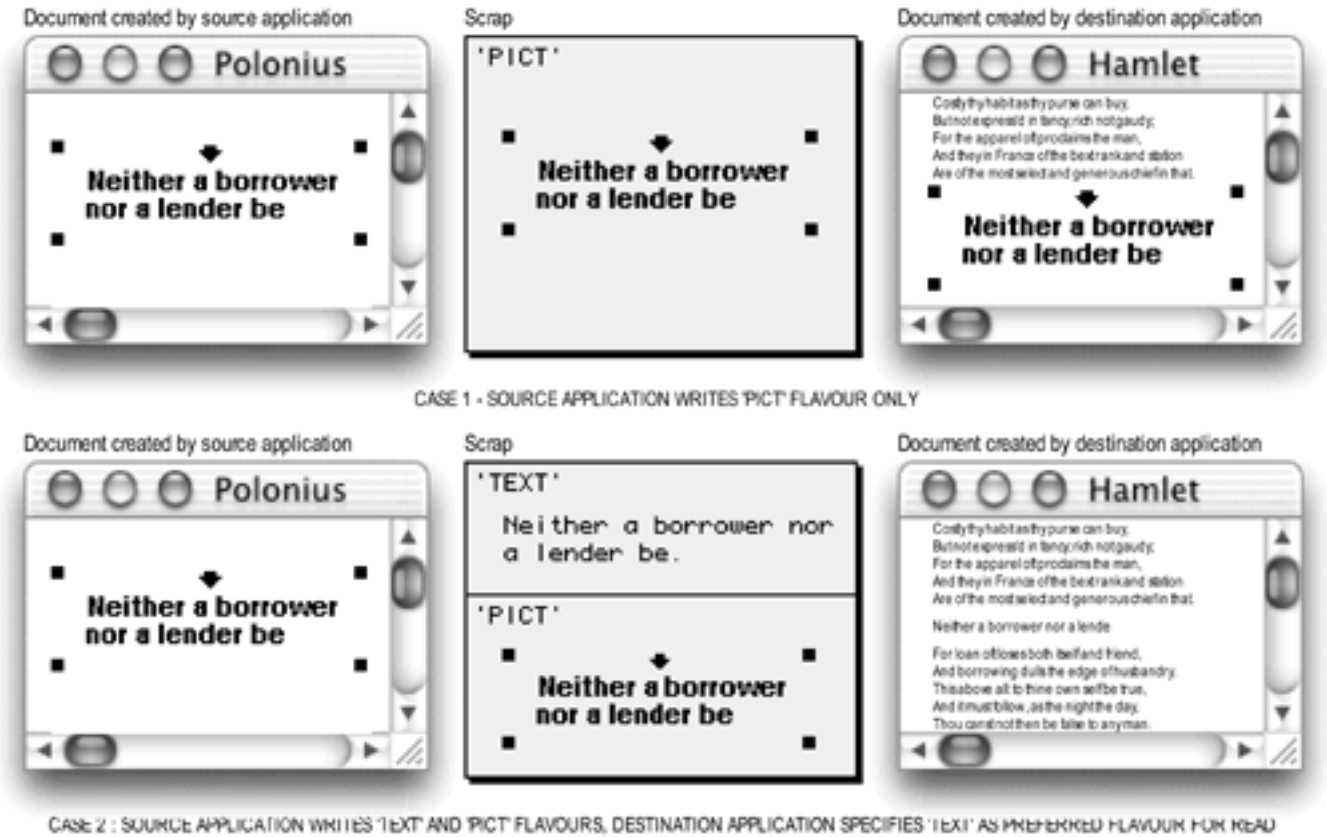
Enabling the Paste Menu Item

Your application can determine whether to enable the Paste item in the Edit menu by calling `GetScrapFlavorFlags` to determine whether the scrap contains data of the flavor type specified in that call. `GetScrapFlavorFlags` returns `noErr` if the specified flavor exists.

Example

Fig 1 illustrates two cases, both of which deal with a user copying a picture consisting of text from a source document created by one application to a destination document created by another application.

Figure 1. SPECIFYING FLAVOURS TO WRITE TO AND READ FROM THE DESK SCRAP


Document created by source application
Polonius
Neither a borrower nor a lender be
Scrap
'PICT'
Neither a borrower nor a lender be
Document created by destination application
Hamlet
Neither a borrower nor a lender be
CASE 1 - SOURCE APPLICATION WRITES 'PICT' FLAVOUR ONLY
Document created by source application
Polonius
Neither a borrower nor a lender be
Scrap
'TEXT'
Neither a borrower nor a lender be.
'PICT'
Neither a borrower nor a lender be
Document created by destination application
Hamlet
Neither a borrower nor a lende
CASE 2 : SOURCE APPLICATION WRITES 'TEXT' AND 'PICT' FLAVOURS, DESTINATION APPLICATION SPECIFIES 'TEXT' AS PREFERRED FLAVOUR FOR READ

In the first case, the source application has chosen to write only the `'PICT'` flavor to the scrap, and the destination application has pasted the data, in that flavor, to its document.

In the second case, the source application has chosen to write both the `'TEXT'` and `'PICT'` flavors to the scrap, and the destination application has chosen the `'TEXT'` flavor as the preferred flavor for the paste. The data is thus inserted into the document as editable text.

Clipboard Windows

Your application can provide a Show Clipboard command in the Edit menu which, when chosen, shows a window that displays the current contents of the scrap. Such a window is known as a **Clipboard** window. The Show Clipboard command should be toggled with a Hide Clipboard command to allow the user to hide the Clipboard window when required.

Although the scrap may contain multiple scrap flavors, your Clipboard window should ordinarily display the data in the application's preferred flavor only.

If the user has chosen to open the Clipboard window, your application should hide the window on receipt of a suspend event (Classic event model) or `kEventAppActivated` event type (Carbon event model) and show it when a resume event (Classic event model) or `kEventAppDeactivated` event type (Carbon event model) is received. This is necessary because the contents of the scrap could change while the application is in the background.

Transferring the Scrap to Disk -Mac OS 8/9

Although, on Mac OS 8/9, the scrap is usually located in memory, your application can write the contents of the scrap in memory to a scrap file using `UnloadScrap`. You should do this only if memory is not large enough to hold the data you need to write to the scrap. After writing the contents of the scrap to disk, `UnloadScrap` releases the memory previously occupied by the scrap. Thereafter, any operations your application performs on data in the scrap affect the scrap as stored in the scrap file on disk. You can use `LoadScrap` to read the contents of the scrap file back into memory.

On Mac OS X, calls to `LoadScrap` and `UnloadScrap` are ignored.

Main Carbon Scrap Manager Functions

The main Carbon Scrap Manager functions are as follows:

Function	*Description*

`GetCurrentScrap`	Gets a reference to the current scrap. (Note that this reference will become invalid and unusable after the scrap is cleared.)
`GetScrapFlavorFlags`	Determines whether the scrap contains data for a particular flavor and provides information about that flavor if it exists. (This function is useful for deciding whether to enable the Paste item in your Edit menu.)
`GetScrapFlavorSize`	Gets the size of the data of the specified flavor from the specified scrap.
`GetScrapFlavorData`	Gets the data of the specified flavor from the specified scrap.
`ClearCurrentScrap`	Clears the current scrap. This function should be called immediately the user requests a Copy or Cut operation.
`PutScrapFlavor`	Puts data on the scrap. Also promises data to the specified scrap (see below).

Associated Constants and Data Types

The following constants and data types are associated with the main Scrap Manager functions:

Scrap Flavor Type Constants

Constant	*Flavor Type*	*Description*
`kScrapFlavorTypePicture`	`'PICT'`	Picture
`kScrapFlavorTypeText`	`'TEXT'`	Text
`kScrapFlavorTypeTextStyle`	`'styl'`	Text style
`kScrapFlavorTypeMovie`	`'moov'`	Movie

Scrap Flavor Flag Constants

In the following, the first two constants may be passed in the `flavorFlags` parameter in calls to `PutScrapFlavor`, and the third is received in the `flavorFlags` parameter in calls to `GetScrapFlavorFlags`:

Constant	*Meaning*
`kScrapFlavorMaskNone`	No flags required.

`kScrapFlavorMaskSenderOnly`	Only the process, which puts the flavor on the scrap, can see it. If another process puts a flavor with this flag on the scrap,your process will never see the flavor. Accordingly, there is no point in testing for this flag. This flag is typically used to save a private flavor to the scrap so that other promised (see below) public flavors can be derived from it on demand.
`kScrapFlavorMaskTranslated`	The flavor was translated, by the Translation Manager, from some other flavor in the scrap. (Most callers should not care about this flag.)

ScrapFlavorInfo Data Type

The `ScrapFlavorInfo` data type describes a single flavor within a scrap and is used by those functions that get information about the current scrap (`GetScrapFlavorFlags`, `GetScrapFlavorSize`, and `GetScrapFlavorData`):

```
struct ScrapFlavorInfo
{
  ScrapFlavorType  flavorType;
  ScrapFlavorFlags flavorFlags;
} ;
typedef struct ScrapFlavorInfo ScrapFlavorInfo;
```

Private Scrap

As an alternative to writing to and reading from the scrap whenever the user cuts, copies, and pastes data, your application can choose to use its own **private scrap.** An application that uses a private scrap copies data to its private scrap when the user chooses the Cut or Copy command and pastes data from the private scrap when the user chooses the Paste command.

Additional Actions-Old Scrap Manager

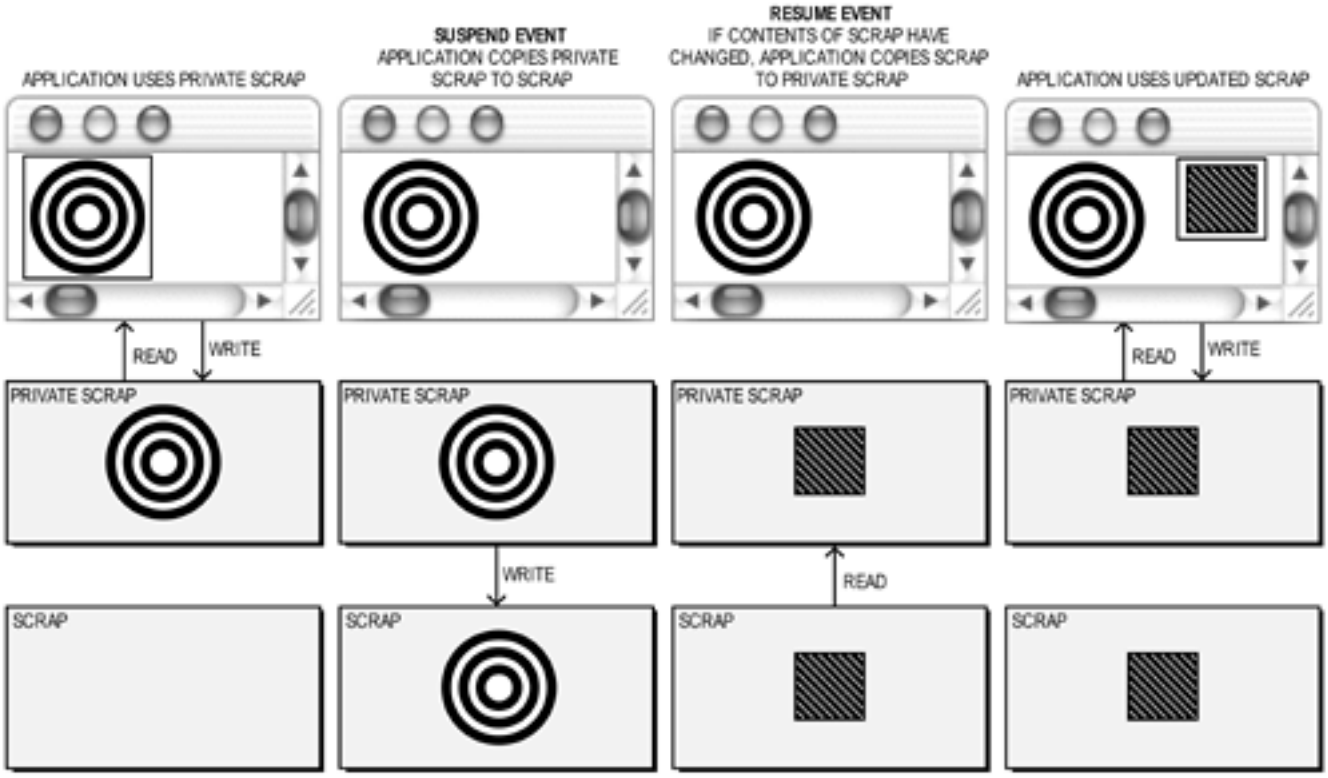
In the old pre-Carbon Scrap Manager, an application that used a private scrap had to take the following additional actions whenever it received suspend and resume events:

- ***Suspend Event.*** On receipt of a suspend event, the application had to copy data from the private scrap to the scrap.

- ***Resume Event.*** On receipt of a resume event, the application had to first examine the `convertClipboardFlag` bit in the `message` field of the resume event structure to determine if the data in the scrap had changed since the previous suspend event. If the data in the scrap had changed, the application had to copy the data from the scrap to its private scrap. The application's menu adjustment function enabled the Paste item if the data copied to the private scrap was of the preferred, or other acceptable, type.

The process is illustrated in Fig 2.

Figure 2. PRIVATE SCRAP AND SUSPEND AND RESUME EVENTS - OLD SCRAP MANAGER



Additional Actions-Carbon Scrap Manager

In the preemptively scheduled Mac OS X, this rather straightforward approach is no longer feasible. Consider the following scenario on Mac OS X:

- Application B, which has a private scrap, is the frontmost application. The user clicks in a window belonging to application A to make application A the frontmost application. Application B receives a suspend event and begins to convert its private scrap.

- While application B is still converting its private scrap, application A has become the frontmost application, and the user clicks in its menu bar. Application A, needing to decide whether to enable the Paste item in its Edit menu, looks at the scrap to determine what flavors it contains. Because application B has not finished converting its private scrap, and thus has not put anything onto the scrap, application A finds nothing it wants on the scrap and, accordingly, disables the Paste item.

The situation in which application A finds itself with regard to the Paste item is not acceptable in terms of human interface. The user cannot be expected to know that application B is still converting its scrap and that application A's Paste item will be enabled in due course.

Making Promises

The Carbon Scrap Manager eliminates this problem using the concept of **promised flavors.** If, in the above example, application B calls `PutScrapFlavor` with `NULL` passed in the `flavorData` parameter whenever the user chooses Cut or Copy, a promise is made that data

of the flavor specified in the `flavorType` parameter will later be placed on the scrap. On checking the scrap, application A will see the promise and can thus enable its Paste item in the expectation that the actual data will eventually appear in the scrap. The actual data can then be provided by application B through a subsequent call to `PutScrapFlavor` during the execution of a **scrap promise keeper (callback) function.** (Scrap promise keeper callback functions are called by the Carbon Scrap Manager as required to keep an earlier promise of a particular scrap flavor.)

In the first (promise-making) call to `PutScrapFlavor`, passing a non-zero size in the `flavorSize` parameter is optional; however, providing the size is advisable because callers of `GetScrapFlavorSize` will then be able to avoid blocking. If the size is provided, the subsequent call to `PutScrapFlavor` must provide the same amount of data as was promised. If the size is unknown at the time of the promise, your application should pass `kScrapFlavorSizeUnknown` in the `flavorSize` parameter.

Note that the promise-making `PutScrapFlavor` call cannot be made when your application receives a suspend event. This is because of the fundamental difference between the receipt of suspend events in Carbon applications as compared with Classic applications (see Chapter 2). Making the promise each time the user chooses Cut or Copy involves very little overhead, since only the promise, not the data, is being placed on the scrap.

Calling In Promises

In applications that use the Classic event model, your application should invariably call `CallInScrapPromises` on exit to cater for the possibility that it may have made promises that, after it quits, it cannot possibly honor. `CallInScrapPromises` forces all promises to be kept. On Mac OS X, this action is necessary even if your application has itself made no promises, the reason being that it is possible that, unbeknown to the application, promises could have been made on its behalf. For example, when you copy `TEXT` data (which has ASCII 13 for line endings) onto the scrap, the Carbon Scrap Manager promises other flavors, which have different line endings and/or text encodings so that Cocoa applications can paste.

Calling `CallInScrapPromises` is not necessary in applications which use the Carbon event model.

TextEdit, Dialogs, and Scrap

TextEdit and Scrap

TextEdit is a collection of functions and data structures that you can use to provide your application with basic text editing capabilities.

If your application uses TextEdit in its windows, be aware that TextEdit maintains its own private scrap. Accordingly

- The special TextEdit functions `TECut`, `TECopy`, and `TEToScrap` are used in the processes of cutting text from the document and copying text to the TextEdit private scrap and to the scrap.

- The special TextEdit functions `TEPaste`, `TEStylePaste`, and `TEFromScrap` are used in the processes of pasting text from the TextEdit private scrap and copying text from the scrap to the TextEdit private scrap.

Chapter 21 describes TextEdit, including the TextEdit private scrap and the TextEdit scrap-related functions.

Dialogs and Scrap

Dialogs may contain edit text items, and the Dialog Manager uses TextEdit to perform the editing operations within those items.

You can use the Dialog Manager to handle most editing operations within dialogs. The Dialog Manager functions `DialogCut`, `DialogCopy`, and `DialogPaste` may be used to implement Cut, Copy and Paste commands within edit text items in dialogs. (See the demonstration program in Chapter 8.)

TextEdit's private scrap facilitates the copying and pasting of data between dialogs. However, your application itself must ensure that the user can copy and paste data between your application's dialogs and its document windows. If your application uses TextEdit for all editing operations within its document windows, this is easily achieved because TextEdit's `TECut`, `TECopy`, `TEPaste`, and `TEStylePaste` functions and the Dialog Manager's `DialogCut`, `DialogCopy`, and `DialogPaste` functions all use TextEdit's private scrap.

Main Carbon Scrap Manager Constants, Data Types, and Functions

Constants

Scrap Flavor Types

```
kScrapFlavorTypePicture     = FOUR_CHAR_CODE('PICT')  Picture
kScrapFlavorTypeText        = FOUR_CHAR_CODE('TEXT')  Text
kScrapFlavorTypeTextStyle   = FOUR_CHAR_CODE('styl')  Text style
kScrapFlavorTypeMovie       = FOUR_CHAR_CODE('moov')  Movie
```

Scrap Flavor Flags

```
kScrapFlavorMaskNone        = 0x00000000
kScrapFlavorMaskSenderOnly  = 0x00000001
kScrapFlavorMaskTranslated  = 0x00000002
```

Promising Flavors

```
kScrapFlavorSizeUnknown     = -1
```

Result Codes

```
internalScrapErr            = -4988
duplicateScrapFlavorErr     = -4989
badScrapRefErr              = -4990
processStateIncorrectErr    = -4991
scrapPromiseNotKeptErr      = -4992
noScrapPromiseKeeperErr     = -4993
nilScrapFlavorDataErr       = -4994
scrapFlavorFlagsMismatchErr = -4995
scrapFlavorSizeMismatchErr  = -4996
illegalScrapFlavorFlagsErr  = -4997
illegalScrapFlavorTypeErr   = -4998
illegalScrapFlavorSizeErr   = -4999
scrapFlavorNotFoundErr      = -102
needClearScrapErr           = -100
```

Data Types

```
typedef struct OpaqueScrapRef  *ScrapRef;
typedef FourCharCode           ScrapFlavorType;
typedef UInt32                 ScrapFlavorFlags;
```

ScrapFlavorInfo

```
struct ScrapFlavorInfo
{
  ScrapFlavorType  flavorType;
  ScrapFlavorFlags flavorFlags;
} ;
typedef struct ScrapFlavorInfo ScrapFlavorInfo;
```

Functions

Obtaining a Reference to the Current Scrap

```
OSStatus  GetCurrentScrap(ScrapRef *scrap);
```

Obtaining Information About a Specific Scrap Flavor

```
OSStatus  GetScrapFlavorFlags(ScrapRef scrap,ScrapFlavorType flavorType,
          ScrapFlavorFlags *flavorFlags);
```

Obtaining the Size of Data of a Specified Scrap Flavor

```
OSStatus  GetScrapFlavorSize(ScrapRef scrap,ScrapFlavorType flavorType,Size *byteCount);
```

Obtaining the Data of a Specified Scrap Flavor

```
OSStatus  GetScrapFlavorData(ScrapRef scrap,ScrapFlavorType flavorType,Size *byteCount,
          void *destination);
```

Writing Data to the Scrap and Clearing the Scrap

```
OSStatus  PutScrapFlavor(ScrapRef scrap,ScrapFlavorType flavorType,
          ScrapFlavorFlags flavorFlags, Size flavorSize,const void *flavorData);
OSStatus  ClearCurrentScrap(void);
```

Scrap Promise Keeping

```
ScrapPromiseKeeperUPP  NewScrapPromiseKeeperUPP(ScrapPromiseKeeperProcPtr userRoutine);
void                     DisposeScrapPromiseKeeperUPP(ScrapPromiseKeeperUPP userUPP);
OSStatus  SetScrapPromiseKeeper(ScrapRef scrap,ScrapPromiseKeeperUPP upp,
          const void *userData);
OSStatus  CallInScrapPromises(void);
```

Application-Defined (Callback) Function

```
OSStatus  myScrapPromiseKeeperFunction(ScrapRef scrap,ScrapFlavorType flavorType,
                                       void *userData);
```

Transferring the Scrap Between Memory and Disk (Mac OS 8/9)

```
SInt32    UnloadScrap(void);  // Does nothing when called on Mac OS X
SInt32    LoadScrap(void);    // Does nothing when called on Mac OS X
```

Demonstration Program CarbonScrap Listing

```
// ****************************************************************************
// CarbonScrap.c                                                 CARBON EVENT MODEL
// ****************************************************************************
//
// This program utilises Carbon Scrap Manager functions to allow the user to:
//
// •  Cut, copy, clear, and paste text and pictures from and to two document windows opened by
//    the program.
//
// •  Paste text and pictures cut or copied from another application to the two document
//    windows.
//
// •  Open and close a Clipboard window, in which the current contents of the scrap are
//    displayed.
//
// The program's preferred scrap flavor type is 'TEXT'. Thus, if the scrap contains data in
// both the 'TEXT' and 'PICT' flavor types, only the 'TEXT' flavor will be used for pastes
// to the document windows and display in the Clipboard window.
//
// In order to keep that part of the source code that is not related to the Carbon Scrap
// Manager to a minimum, the windows do not display insertion points, nor can the pictures be
// dragged within the windows. The text and pictures are not inserted into a document as
// such. Rather, when the Paste item in the Edit menu is chosen:
//
// •  The text or picture on the Clipboard is simply drawn in the center of the active window.
//
// •  A handle to the text or picture is assigned to fields in a document structure associated
//    with the window. (The demonstration program MonoTextEdit (Chapter 21) shows how to cut,
//    copy, and paste text from and to a TextEdit structure using the scrap.)
//
```

```
// For the same reason, highlighting the selected text or picture in a window is simplified by
// simply inverting the image.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  An 'MBAR' resource, and 'MENU' resources for Apple, File, and Edit menus (preload,
//    non-purgeable).
//
// •  A 'TEXT' resource (non-purgeable) containing text displayed in the left window at
//    program  start.
//
// •  A 'PICT' resource (non-purgeable) containing a picture displayed in the right window at
//    program start.
//
// •  A 'STR#' resource (purgeable) containing strings to be displayed in the error Alert.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// ************************************************************************************

// ..................................................................... includes

#include <Carbon.h>

// ...................................................................... defines

#define rMenubar          128
#define mAppleApplication 128
#define  iAbout           1
#define mFile             129
#define  iClose           4
#define  iQuit            12
```

```
#define mEdit             130
#define  iCut             3
#define  iCopy            4
#define  iPaste           5
#define  iClear           6
#define  iClipboard       8
#define rText             128
#define rPicture          128
#define rErrorStrings     128
#define  eFailMenu        1
#define  eFailWindow      2
#define  eFailDocStruc    3
#define  eFailMemory      4
#define  eClearScrap      5
#define  ePutScrapFlavor  6
#define  eGetScrapSize    7
#define  eGetScrapData    8
#define kDocumentType     1
#define kClipboardType    2
#define MAX_UINT32        0xFFFFFFFF

// ..................................................................... typedefs

typedef struct
{
  PicHandle pictureHdl;
  Handle    textHdl;
  Boolean   selectFlag;
  SInt16    windowType;
}  docStructure, **docStructureHandle;

// ................................................................ global variables

Boolean   gRunningOnX        = false;
WindowRef gClipboardWindowRef = NULL;
```

```
Boolean   gClipboardShowing   = false;

// ..................................................................... function prototypes

void            main                   (void);
void            doPreliminaries        (void);
OSStatus        appEventHandler        (EventHandlerCallRef,EventRef,void *);
OSStatus        docWindowEventHandler  (EventHandlerCallRef,EventRef,void *);
OSStatus        clipWindowEventHandler (EventHandlerCallRef,EventRef,void *);
void            doAdjustMenus          (void);
void            doMenuChoice           (MenuID,MenuItemIndex);
void            doErrorAlert           (SInt16);
void            doOpenDocumentWindows  (void);
EventHandlerUPP doGetHandlerUPP        (void);
void            doCloseWindow          (void);
void            doInContent            (Point);
void            doCutCopyCommand       (Boolean);
void            doPasteCommand         (void);
void            doClearCommand         (void);
void            doClipboardCommand     (void);
void            doDrawClipboardWindow  (void);
void            doDrawDocumentWindow   (WindowRef);
Rect            doSetDestRect          (Rect *,WindowRef);

// ********************************************************************************** main

void  main(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  EventTypeSpec applicationEvents[] = { { kEventClassApplication, kEventAppActivated    } ,
                                        { kEventClassApplication, kEventAppDeactivated } ,
                                        { kEventClassCommand,     kEventProcessCommand } ,
                                        { kEventClassMenu,        kEventMenuEnableItems } } ;
```

```
// ..................................................................... do preliminaries

doPreliminaries();

// ..................................................................... set up menu bar and menus

menubarHdl = GetNewMBar(rMenubar);
if(menubarHdl == NULL)
  doErrorAlert(eFailMenu);
SetMenuBar(menubarHdl);
DrawMenuBar();

Gestalt(gestaltMenuMgrAttr,&response);
if(response & gestaltMenuMgrAquaLayoutMask)
{
  menuRef = GetMenuRef(mFile);
  if(menuRef != NULL)
  {
    DeleteMenuItem(menuRef,iQuit);
    DeleteMenuItem(menuRef,iQuit - 1);
    DisableMenuItem(menuRef,0);
  }

  gRunningOnX = true;
}
else
{
  menuRef = GetMenuRef(mFile);
  if(menuRef != NULL)
    SetMenuItemCommandID(menuRef,iQuit,kHICommandQuit);
}

// ..................................................................... open document windows
```

```
  doOpenDocumentWindows();
  // ..................................................... install application event handler

  InstallApplicationEventHandler(NewEventHandlerUPP((EventHandlerProcPtr) appEventHandler),
                                 GetEventTypeCount(applicationEvents),applicationEvents,
                                 0,NULL);

  // ..................................................... run application event loop

  RunApplicationEventLoop();
}

// ********************************************************************** doPreliminaries

void  doPreliminaries(void)
{
  MoreMasterPointers(96);
  InitCursor();
}

// ********************************************************************** appEventHandler

OSStatus  appEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                          void * userData)
{
  OSStatus      result = eventNotHandledErr;
  UInt32        eventClass;
  UInt32        eventKind;
  HICommand     hiCommand;
  MenuID        menuID;
  MenuItemIndex menuItem;

  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);
```

```
switch(eventClass)
{
  case kEventClassApplication:
    if(eventKind == kEventAppActivated)
    {
      SetThemeCursor(kThemeArrowCursor);
      if(gClipboardWindowRef && gClipboardShowing)
        ShowWindow(gClipboardWindowRef);
    }
    else if(eventKind == kEventAppDeactivated)
    {
      if(gClipboardWindowRef && gClipboardShowing)
        ShowHide(gClipboardWindowRef,false);
    }
  break;

  case kEventClassCommand:
    if(eventKind == kEventProcessCommand)
    {
      GetEventParameter(eventRef,kEventParamDirectObject,typeHICommand,NULL,
                        sizeof(HICommand),NULL,&hiCommand);
      menuID = GetMenuID(hiCommand.menu.menuRef);
      menuItem = hiCommand.menu.menuItemIndex;
      if((hiCommand.commandID != kHICommandQuit) &&
         (menuID >= mAppleApplication && menuID <= mEdit))
      {
        doMenuChoice(menuID,menuItem);
        result = noErr;
      }
    }
    break;

  case kEventClassMenu:
    if(eventKind == kEventMenuEnableItems)
      doAdjustMenus();
```

```
      result = noErr;
      break;
  }

  return result;
}

// ********************************************************************** docWindowEventHandler

OSStatus  docWindowEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                                void * userData)
{
  OSStatus           result = eventNotHandledErr;
  UInt32             eventClass;
  UInt32             eventKind;
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  Point              mouseLocation;

  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);

  switch(eventClass)
  {
    case kEventClassWindow:
      GetEventParameter(eventRef,kEventParamDirectObject,typeWindowRef,NULL,sizeof(windowRef),
                    NULL,&windowRef);
      switch(eventKind)
      {
        case kEventWindowDrawContent:
          docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
          if((*docStrucHdl)->pictureHdl != NULL || (*docStrucHdl)->textHdl != NULL)
            doDrawDocumentWindow(windowRef);
          result = noErr;
          break;
```

```
        case kEventWindowClickContentRgn:
          GetEventParameter(eventRef,kEventParamMouseLocation,typeQDPoint,NULL,
                            sizeof(mouseLocation),NULL,&mouseLocation);
          SetPortWindowPort(windowRef);
          GlobalToLocal(&mouseLocation);
          doInContent(mouseLocation);
          result = noErr;
          break;
      }
      break;
  }

  return result;
}

// ********************************************************************** clipWindowEventHandler

OSStatus  clipWindowEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                                 void * userData)
{
  OSStatus result = eventNotHandledErr;
  UInt32   eventClass;
  UInt32   eventKind;
  MenuRef  editMenuRef;

  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);

  if(eventClass == kEventClassWindow)
  {
    switch(eventKind)
    {
      case kEventWindowActivated:
      case kEventWindowDeactivated:
      case kEventWindowDrawContent:
```

```
        doDrawClipboardWindow();
        result = noErr;
        break;

      case kEventWindowClose:
        DisposeWindow(gClipboardWindowRef);
        gClipboardWindowRef = NULL;
        gClipboardShowing = false;
        editMenuRef = GetMenuRef(mEdit);
        SetMenuItemText(editMenuRef,iClipboard,"\pShow Clipboard");
        break;
    }
  }

  return result;
}
// ********************************************************************************* doAdjustMenus

void  doAdjustMenus(void)
{
  MenuRef            fileMenuRef, editMenuRef;
  docStructureHandle docStrucHdl;
  ScrapRef           scrapRef;
  OSStatus           osError;
  ScrapFlavorFlags   scrapFlavorFlags;
  Boolean            scrapHasText = false, scrapHasPicture = false;

  fileMenuRef = GetMenuRef(mFile);
  editMenuRef = GetMenuRef(mEdit);

  docStrucHdl = (docStructureHandle) GetWRefCon(FrontWindow());

  if((*docStrucHdl)->windowType == kClipboardType)
    EnableMenuItem(fileMenuRef,iClose);
  else
```

```
    DisableMenuItem(fileMenuRef,iClose);

  if(((*docStrucHdl)->pictureHdl || (*docStrucHdl)->textHdl) && ((*docStrucHdl)->selectFlag))
  {
    EnableMenuItem(editMenuRef,iCut);
    EnableMenuItem(editMenuRef,iCopy);
    EnableMenuItem(editMenuRef,iClear);
  }
  else
  {
    DisableMenuItem(editMenuRef,iCut);
    DisableMenuItem(editMenuRef,iCopy);
    DisableMenuItem(editMenuRef,iClear);
  }

  GetCurrentScrap(&scrapRef);

  osError = GetScrapFlavorFlags(scrapRef,kScrapFlavorTypeText,&scrapFlavorFlags);
  if(osError == noErr)
    scrapHasText = true;

  osError = GetScrapFlavorFlags(scrapRef,kScrapFlavorTypePicture,&scrapFlavorFlags);
  if(osError == noErr)
    scrapHasPicture = true;

  if((scrapHasText || scrapHasPicture) && ((*docStrucHdl)->windowType != kClipboardType))
    EnableMenuItem(editMenuRef,iPaste);
  else
    DisableMenuItem(editMenuRef,iPaste);

  DrawMenuBar();
}
// ********************************************************************** doMenuChoice

void  doMenuChoice(MenuID menuID,MenuItemIndex menuItem)
```

```
{
  if(menuID == 0)
    return;

  switch(menuID)
  {
    case mAppleApplication:
      if(menuItem == iAbout)
        SysBeep(10);
      break;

    case mFile:
      if(menuItem == iClose)
        doCloseWindow();
      break;

    case mEdit:
      switch(menuItem)
      {
        case iCut:
          doCutCopyCommand(true);
          break;

        case iCopy:
          doCutCopyCommand(false);
          break;

        case iPaste:
          doPasteCommand();
          break;

        case iClear:
          doClearCommand();
          break;
```

```
        case iClipboard:
          doClipboardCommand();
          break;
      }
      break;
  }
}

// ********************************************************************************** doErrorAlert

void  doErrorAlert(SInt16 errorCode)
{
  Str255  errorString;
  SInt16  itemHit;
  GetIndString(errorString,rErrorStrings,errorCode);
  StandardAlert(kAlertStopAlert,errorString,NULL,NULL,&itemHit);
  ExitToShell();
}

// ************************************************************************ doOpenDocumentWindows

void  doOpenDocumentWindows(void)
{
  SInt16            a;
  OSStatus          osError;
  WindowRef         windowRef;
  Rect              contentRect = { 43,7,223,297 } , theRect;
  Str255            title1      = "\pDocument A";
  Str255            title2      = "\pDocument B";
  docStructureHandle docStrucHdl;
  EventTypeSpec     windowEvents[] = {  {  kEventClassWindow,  kEventWindowDrawContent     } ,
                                        {  kEventClassWindow,  kEventWindowClickContentRgn }  } ;

  for(a=0;a<2;a++)
  {
```

```
osError = CreateNewWindow(kDocumentWindowClass,kWindowStandardHandlerAttribute,
                          &contentRect,&windowRef);
if(osError != noErr)
  doErrorAlert(eFailWindow);

if(a == 0)
{
  SetWTitle(windowRef,"\pDocument A");
  OffsetRect(&contentRect,305,0);
}
else
  SetWTitle(windowRef,"\pDocument B");

if(!(docStrucHdl = (docStructureHandle) NewHandle(sizeof(docStructure))))
  doErrorAlert(eFailDocStruc);
SetWRefCon(windowRef,(SInt32) docStrucHdl);

(*docStrucHdl)->pictureHdl = NULL;
(*docStrucHdl)->textHdl    = NULL;
(*docStrucHdl)->windowType = kDocumentType;
(*docStrucHdl)->selectFlag = false;

SetPortWindowPort(windowRef);

if(gRunningOnX)
{
  GetWindowPortBounds(windowRef,&theRect);
  InsetRect(&theRect,40,40);
  ClipRect(&theRect);
}
else
  UseThemeFont(kThemeSmallSystemFont,smSystemScript);
if(a == 0)
  (*docStrucHdl)->textHdl = (Handle) GetResource('TEXT',rText);
else
```

```
      (*docStrucHdl)->pictureHdl = GetPicture(rPicture);

    InstallWindowEventHandler(windowRef,doGetHandlerUPP(),GetEventTypeCount(windowEvents),
                              windowEvents,0,NULL);

    ShowWindow(windowRef);
  }
}

// ********************************************************************** doGetHandlerUPP

EventHandlerUPP  doGetHandlerUPP(void)
{
  static EventHandlerUPP  windowEventHandlerUPP;

  if(windowEventHandlerUPP == NULL)
    windowEventHandlerUPP = NewEventHandlerUPP((EventHandlerProcPtr) docWindowEventHandler);

  return windowEventHandlerUPP;
}

// ********************************************************************** doCloseWindow

void  doCloseWindow(void)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  MenuRef            editMenuRef;

  windowRef = FrontWindow();
  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

  if((*docStrucHdl)->windowType == kClipboardType)
  {
    DisposeWindow(windowRef);
```

```
    gClipboardWindowRef = NULL;
    gClipboardShowing = false;
    editMenuRef = GetMenuRef(mEdit);
    SetMenuItemText(editMenuRef,iClipboard,"\pShow Clipboard");
  }
}

// ********************************************************************************** doInContent

void  doInContent(Point mouseLocation)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  GrafPtr            oldPort;
  Rect               theRect;
  windowRef = FrontWindow();
  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

  if((*docStrucHdl)->windowType == kClipboardType)
    return;

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

  if((*docStrucHdl)->textHdl != NULL || (*docStrucHdl)->pictureHdl != NULL)
  {
    if((*docStrucHdl)->textHdl != NULL)
    {
      GetWindowPortBounds(windowRef,&theRect);
      InsetRect(&theRect,40,40);
    }
    else if((*docStrucHdl)->pictureHdl != NULL)
    {
      theRect = doSetDestRect(&(*(*docStrucHdl)->pictureHdl)->picFrame,windowRef);
    }
```

```
    if(PtInRect(mouseLocation,&theRect) && (*docStrucHdl)->selectFlag == false)
    {
      (*docStrucHdl)->selectFlag = true;
      InvertRect(&theRect);
    }
    else if(!PtInRect(mouseLocation,&theRect) && (*docStrucHdl)->selectFlag == true)
    {
      (*docStrucHdl)->selectFlag = false;
      InvertRect(&theRect);
    }
  }

  SetPort(oldPort);
}

// ********************************************************************** doCutCopyCommand

void  doCutCopyCommand(Boolean cutFlag)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  OSStatus           osError;
  ScrapRef           scrapRef;
  Size               dataSize;
  GrafPtr            oldPort;
  Rect               portRect;

  windowRef = FrontWindow();
  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

  if((*docStrucHdl)->selectFlag == false)
    return;
  osError = ClearCurrentScrap();
  if(osError == noErr)
```

```
{
  GetCurrentScrap(&scrapRef);

  if((*docStrucHdl)->textHdl != NULL)  // ..........................................'TEXT'
  {
    dataSize = GetHandleSize((Handle) (*docStrucHdl)->textHdl);
    HLock((*docStrucHdl)->textHdl);

    osError = PutScrapFlavor(scrapRef,kScrapFlavorTypeText,kScrapFlavorMaskNone,
                             dataSize,*((*docStrucHdl)->textHdl));
    if(osError != noErr)
      doErrorAlert(ePutScrapFlavor);
  }
  else if((*docStrucHdl)->pictureHdl != NULL)  // ..................................... 'PICT'
  {
    dataSize = GetHandleSize((Handle) (*docStrucHdl)->pictureHdl);
    HLock((Handle) (*docStrucHdl)->pictureHdl);

    osError =PutScrapFlavor(scrapRef,kScrapFlavorTypePicture,kScrapFlavorMaskNone,
                             dataSize,*((Handle) (*docStrucHdl)->pictureHdl));
    if(osError != noErr)
      doErrorAlert(ePutScrapFlavor);
  }

  if((*docStrucHdl)->textHdl != NULL)
    HUnlock((*docStrucHdl)->textHdl);
  if((*docStrucHdl)->pictureHdl != NULL)
    HUnlock((Handle) (*docStrucHdl)->pictureHdl);
}
else
  doErrorAlert(eClearScrap);

if(cutFlag)
{
  GetPort(&oldPort);
```

```
    SetPortWindowPort(windowRef);

    if((*docStrucHdl)->pictureHdl != NULL)
    {
      DisposeHandle((Handle) (*docStrucHdl)->pictureHdl);
      (*docStrucHdl)->pictureHdl = NULL;
      (*docStrucHdl)->selectFlag = false;
    }
    if((*docStrucHdl)->textHdl != NULL)
    {
      DisposeHandle((*docStrucHdl)->textHdl);
      (*docStrucHdl)->textHdl = NULL;
      (*docStrucHdl)->selectFlag = false;
    }

    GetWindowPortBounds(windowRef,&portRect);
    EraseRect(&portRect);
    SetPort(oldPort);
  }

  if(gClipboardWindowRef != NULL)
    doDrawClipboardWindow();
}

// ********************************************************************** doPasteCommand

void  doPasteCommand(void)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  GrafPtr            oldPort;
  ScrapRef           scrapRef;
  OSStatus           osError;
  ScrapFlavorFlags   flavorFlags;
  Size               sizeOfPictData = 0, sizeOfTextData = 0;
```

```
Handle              newTextHdl, newPictHdl;
CFStringRef         stringRef;
Rect                destRect, portRect;

windowRef = FrontWindow();
docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

GetPort(&oldPort);
SetPortWindowPort(windowRef);

GetCurrentScrap(&scrapRef);

// ..................................................................... 'TEXT'
osError = GetScrapFlavorFlags(scrapRef,kScrapFlavorTypeText,&flavorFlags);
if(osError == noErr)
{
  osError = GetScrapFlavorSize(scrapRef,kScrapFlavorTypeText,&sizeOfTextData);
  if(osError == noErr && sizeOfTextData > 0)
  {
    newTextHdl = NewHandle(sizeOfTextData);
    osError = MemError();
    if(osError == memFullErr)
      doErrorAlert(eFailMemory);

    HLock(newTextHdl);

    osError = GetScrapFlavorData(scrapRef,kScrapFlavorTypeText,&sizeOfTextData,*newTextHdl);
    if(osError != noErr)
      doErrorAlert(eGetScrapData);

    // ..................................................... draw text in window

    GetWindowPortBounds(windowRef,&portRect);
    EraseRect(&portRect);
    InsetRect(&portRect,40,40);
```

```
    if(!gRunningOnX)
    {
      TETextBox(*newTextHdl,sizeOfTextData,&portRect,teFlushLeft);
    }
    else
    {
      stringRef =  CFStringCreateWithBytes(NULL,(UInt8 *) *newTextHdl,sizeOfTextData,
                                           smSystemScript,false);
      DrawThemeTextBox(stringRef,kThemeSmallSystemFont,kThemeStateActive,true,&portRect,
                       teFlushLeft,NULL);
      if(stringRef != NULL)
        CFRelease(stringRef);
    }

    HUnlock(newTextHdl);

    (*docStrucHdl)->selectFlag = false;

    // ........................... assign handle to new text to window's document structure

    if((*docStrucHdl)->textHdl != NULL)
      DisposeHandle((*docStrucHdl)->textHdl);
    (*docStrucHdl)->textHdl = newTextHdl;

    if((*docStrucHdl)->pictureHdl != NULL)
      DisposeHandle((Handle) (*docStrucHdl)->pictureHdl);
    (*docStrucHdl)->pictureHdl = NULL;
  }
  else
    doErrorAlert(eGetScrapSize);
}

// ..................................................................... ' PICT'

else
```

```
{
  (osError = GetScrapFlavorFlags(scrapRef,kScrapFlavorTypePicture,&flavorFlags));
  if(osError == noErr)
  {
    osError = GetScrapFlavorSize(scrapRef,kScrapFlavorTypePicture,&sizeOfPictData);
    if(osError == noErr && sizeOfPictData > 0)
    {
      newPictHdl = NewHandle(sizeOfPictData);
      osError = MemError();
      if(osError == memFullErr)
        doErrorAlert(eFailMemory);

      HLock(newPictHdl);

      osError = GetScrapFlavorData(scrapRef,kScrapFlavorTypePicture,&sizeOfPictData,
                                   *newPictHdl);
      if(osError != noErr)
        doErrorAlert(eGetScrapData);
      // ..................................................... draw picture in window

      GetWindowPortBounds(windowRef,&portRect);
      EraseRect(&portRect);
      (*docStrucHdl)->selectFlag = false;
      destRect = doSetDestRect(&(*(PicHandle) newPictHdl)->picFrame,windowRef);
      DrawPicture((PicHandle) newPictHdl,&destRect);

      HUnlock(newPictHdl);

      (*docStrucHdl)->selectFlag = false;

      // ..................... assign handle to new picture to window's document structure

      if((*docStrucHdl)->pictureHdl != NULL)
        DisposeHandle((Handle) (*docStrucHdl)->pictureHdl);
      (*docStrucHdl)->pictureHdl = (PicHandle) newPictHdl;
```

```
        if((*docStrucHdl)->textHdl != NULL)
          DisposeHandle((Handle) (*docStrucHdl)->textHdl);
        (*docStrucHdl)->textHdl = NULL;
      }
      else
        doErrorAlert(eGetScrapSize);
    }
  }

  SetPort(oldPort);
}

// ********************************************************************** doClearCommand

void  doClearCommand(void)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  GrafPtr            oldPort;
  Rect               portRect;

  windowRef = FrontWindow();
  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

  if((*docStrucHdl)->textHdl != NULL)
  {
    DisposeHandle((*docStrucHdl)->textHdl);
    (*docStrucHdl)->textHdl = NULL;
  }
  if((*docStrucHdl)->pictureHdl != NULL)
```

```
  {
    DisposeHandle((Handle) (*docStrucHdl)->pictureHdl);
    (*docStrucHdl)->pictureHdl = NULL;
  }

  (*docStrucHdl)->selectFlag = false;

  GetWindowPortBounds(windowRef,&portRect);
  EraseRect(&portRect);

  SetPort(oldPort);
}

// ************************************************************************ doClipboardCommand

void  doClipboardCommand(void)
{
  MenuRef            editMenuRef;
  OSStatus           osError;
  Rect               contentRect = { 254,7,384,603 } ;
  docStructureHandle docStrucHdl;
  EventTypeSpec      windowEvents[] = { { kEventClassWindow, kEventWindowActivated   } ,
                                        { kEventClassWindow, kEventWindowDeactivated } ,
                                        { kEventClassWindow, kEventWindowDrawContent } ,
                                        { kEventClassWindow, kEventWindowClose       }  } ;
  editMenuRef = GetMenuRef(mEdit);

  if(gClipboardWindowRef == NULL)
  {
    osError = CreateNewWindow(kDocumentWindowClass,kWindowStandardHandlerAttribute |
                              kWindowCloseBoxAttribute,&contentRect,&gClipboardWindowRef);
    if(osError != noErr)
      doErrorAlert(eFailWindow);

    SetWTitle(gClipboardWindowRef,"\pClipboard");
```

```
    SetPortWindowPort(gClipboardWindowRef);

    if(!(docStrucHdl = (docStructureHandle) NewHandle(sizeof(docStructure))))
      doErrorAlert(eFailDocStruc);

    SetWRefCon(gClipboardWindowRef,(SInt32) docStrucHdl);
    (*docStrucHdl)->windowType = kClipboardType;

    SetMenuItemText(editMenuRef,iClipboard,"\pHide Clipboard");

    InstallWindowEventHandler(gClipboardWindowRef,
                          NewEventHandlerUPP((EventHandlerProcPtr) clipWindowEventHandler),
                          GetEventTypeCount(windowEvents),windowEvents,0,NULL);

    ShowWindow(gClipboardWindowRef);
    gClipboardShowing = true;
  }
  else
  {
    if(gClipboardShowing)
    {
      HideWindow(gClipboardWindowRef);
      gClipboardShowing = false;
      SetMenuItemText(editMenuRef,iClipboard,"\pShow Clipboard");
    }
    else
    {
      ShowWindow(gClipboardWindowRef);
      gClipboardShowing = true;
      SetMenuItemText(editMenuRef,iClipboard,"\pHide Clipboard");
    }
  }
}

// ********************************************************************* doDrawClipboardWindow
```

```
void  doDrawClipboardWindow(void)
{
  GrafPtr          oldPort;
  Rect             theRect, textBoxRect;
  ScrapRef         scrapRef;
  OSStatus         osError;
  ScrapFlavorFlags flavorFlags;
  CFStringRef      stringRef;
  Handle           tempHdl;
  Size             sizeOfPictData = 0, sizeOfTextData = 0;
  RGBColor         blackColor = {  0x0000, 0x0000, 0x0000 } ;

  GetPort(&oldPort);
  SetPortWindowPort(gClipboardWindowRef);

  GetWindowPortBounds(gClipboardWindowRef,&theRect);
  EraseRect(&theRect);

  SetRect(&theRect,-1,-1,597,24);
  DrawThemeWindowHeader(&theRect,gClipboardWindowRef == FrontWindow());

  if(gClipboardWindowRef == FrontWindow())
    TextMode(srcOr);
  else
    TextMode(grayishTextOr);

  SetRect(&textBoxRect,10,5,120,20);
  DrawThemeTextBox(CFSTR("Clipboard Contents:"),kThemeSmallSystemFont,0,true,&textBoxRect,
                   teJustLeft,NULL);

  GetCurrentScrap(&scrapRef);
  // ..................................................................................... 'TEXT'

  osError = GetScrapFlavorFlags(scrapRef,kScrapFlavorTypeText,&flavorFlags);
```

```
if(osError == noErr)
{
  osError = GetScrapFlavorSize(scrapRef,kScrapFlavorTypeText,&sizeOfTextData);
  if(osError == noErr && sizeOfTextData > 0)
  {
    SetRect(&textBoxRect,120,5,597,20);
    DrawThemeTextBox(CFSTR("Text"),kThemeSmallSystemFont,0,true,&textBoxRect,teJustLeft,
                     NULL);

    tempHdl = NewHandle(sizeOfTextData);
    osError = MemError();
    if(osError == memFullErr)
      doErrorAlert(eFailMemory);

    HLock(tempHdl);

    osError = GetScrapFlavorData(scrapRef,kScrapFlavorTypeText,&sizeOfTextData,*tempHdl);
    if(osError != noErr)
      doErrorAlert(eGetScrapData);

    // ..................................................... draw text in clipboard window

    GetWindowPortBounds(gClipboardWindowRef,&theRect);
    theRect.top += 22;
    InsetRect(&theRect,2,2);

    if(sizeOfTextData >= 965)
      sizeOfTextData = 965;
    stringRef =  CFStringCreateWithBytes(NULL,(UInt8 *) *tempHdl,sizeOfTextData,
                                         CFStringGetSystemEncoding(),true);
    DrawThemeTextBox(stringRef,kThemeSmallSystemFont,0,true,&theRect,teFlushLeft,NULL);
    if(stringRef != NULL)
      CFRelease(stringRef);

    HUnlock(tempHdl);
```

```
    DisposeHandle(tempHdl);
  }
  else
    doErrorAlert(eGetScrapSize);
}

// ..........................................................................'PICT'

else
{
  osError = GetScrapFlavorFlags(scrapRef,kScrapFlavorTypePicture,&flavorFlags);
  if(osError == noErr)
  {
    osError = GetScrapFlavorSize(scrapRef,kScrapFlavorTypePicture,&sizeOfPictData);
    if(osError == noErr && sizeOfPictData > 0)
    {
      SetRect(&textBoxRect,120,5,597,20);
      DrawThemeTextBox(CFSTR("Picture"),kThemeSmallSystemFont,0,true,&textBoxRect,
                       teJustLeft,NULL);

      tempHdl = NewHandle(sizeOfPictData);
      osError = MemError();
      if(osError == memFullErr)
        doErrorAlert(eFailMemory);

      HLock(tempHdl);

      osError = GetScrapFlavorData(scrapRef,kScrapFlavorTypePicture,&sizeOfPictData,
                                   *tempHdl);
      if(osError != noErr)
        doErrorAlert(eGetScrapData);

      // ..................................................... draw picture in clipboard window

      theRect = (*(PicHandle) tempHdl)->picFrame;
```

```
        OffsetRect(&theRect,-((*(PicHandle) tempHdl)->picFrame.left - 2),
                            -((*(PicHandle) tempHdl)->picFrame.top - 26));
        DrawPicture((PicHandle) tempHdl,&theRect);

        HUnlock(tempHdl);
        DisposeHandle(tempHdl);
      }
      else
        doErrorAlert(eGetScrapSize);
    }
  }

  TextMode(srcOr);
  SetPort(oldPort);
}

// ********************************************************************** doDrawDocumentWindow

void  doDrawDocumentWindow(WindowRef windowRef)
{
  GrafPtr            oldPort;
  docStructureHandle docStrucHdl;
  Rect               destRect;
  CFStringRef        stringRef;

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

  if((*docStrucHdl)->textHdl != NULL)
  {
    GetWindowPortBounds(windowRef,&destRect);
    EraseRect(&destRect);
    InsetRect(&destRect,40,40);
```

```
    stringRef =  CFStringCreateWithBytes(NULL,(UInt8 *) *(*docStrucHdl)->textHdl,
                                         GetHandleSize((*docStrucHdl)->textHdl),
                                         smSystemScript,false);
    DrawThemeTextBox(stringRef,kThemeSmallSystemFont,0,true,&destRect,teFlushLeft,NULL);
    if(stringRef != NULL)
      CFRelease(stringRef);

    if((*docStrucHdl)->selectFlag)
      InvertRect(&destRect);
  }
  else if((*docStrucHdl)->pictureHdl != NULL)
  {
    destRect = doSetDestRect(&(*(*docStrucHdl)->pictureHdl)->picFrame,windowRef);
    DrawPicture((*docStrucHdl)->pictureHdl,&destRect);
    if((*docStrucHdl)->selectFlag)
      InvertRect(&destRect);
  }

  SetPort(oldPort);
}

// ********************************************************************** doSetDestRect

Rect  doSetDestRect(Rect *picFrame,WindowRef windowRef)
{
  Rect   destRect, portRect;
  SInt16 diffX, diffY;

  destRect = *picFrame;
  GetWindowPortBounds(windowRef,&portRect);

  OffsetRect(&destRect,-(*picFrame).left,-(*picFrame).top);

  diffX = (portRect.right - portRect.left) - ((*picFrame).right - (*picFrame).left);
  diffY = (portRect.bottom - portRect.top) - ((*picFrame).bottom - (*picFrame).top);
```

```
  OffsetRect(&destRect,diffX / 2,diffY / 2);

  return destRect;
}

// ****************************************************************************************
```

Demonstration Program CarbonScrap Comments

When this program is run, the user should choose the Edit menu's Show Clipboard item to open the Clipboard window. The user should then cut, copy, clear, and paste the supplied text or picture from/to the two document windows opened by the program, noting the effect on the scrap as displayed in the Clipboard window. (To indicate selection, the text or picture inverts when the user clicks on it with the mouse. The text and picture can be deselected by clicking outside their boundaries.)

The user should also copy text and pictures from another application's window, observing the changes to the contents of the Clipboard window when the demonstration program is brought to the front, and paste that text and those pictures to the document windows. (On Mac OS 8/9, a simple way to get a picture into the scrap is to use Command-Shift-Control-4 to copy an area of the screen to the scrap.)

The program's preferred scrap flavor type is 'TEXT'. Thus, if the scrap contains data in both the 'TEXT' and 'PICT' flavor types, only the 'TEXT' flavor will be used for pastes to the document windows and for display in the Clipboard window. The user can prove this behavior by copying a picture object containing text in an application such as Adobe Illustrator, bringing the demonstration program to the front, noting the contents of the Clipboard window, pasting to one of the document windows, and noting what is pasted.

The user should note that, when the Clipboard window is open and showing, it will be hidden when the program is sent to the background and shown again when the program is brought to the foreground.

defines

`kDocumentType` and `kClipboardType` will enable the program to distinguish between the "document" windows opened by the program and the Clipboard window.

typedefs

Document structures will be attached to the two document windows and the Clipboard window. `docStructure` is the associated data type. The `windowType` field will be used to differentiate between the document windows and the Clipboard window.

Global Variables

`gClipBoardWindowRef` will be assigned a reference to the Clipboard window when it is opened by the user. `gClipBoardShowing` will keep track of whether the Clipboard

window is currently hidden or showing.

appEventHandler

When the `kEventAppActivated` event type is received, if the Clipboard window has been opened and was showing when the program was sent to the background, ShowWindow is called to show the Clipboard window. When the `kEventAppActivated` event type is received, if the Clipboard window has been opened and is currently showing, `ShowHide` is called to hide the Clipboard window. `ShowHide`, rather than `HideWindow` is used in this instance to prevent activation of the first document window in the list when the Clipboard window is in front and the application is switched out.

windowEventHandler

`windowEventHandler` is the handler for the document windows. When the `kEventWindowClickContentRegion` event type is received, the function `doInContent` is called.

clipWindowEventHandler

`clipWindowEventHandler` is the handler for the Clipboard window. The function `doDrawClipboardWindow` is called when the window receives the event types `kEventWindowActivated`, `kEventWindowDeactivated`, `kEventWindowDrawContent` are received. When the `kEventWindowClose` event type is received, the Clipboard window is disposed of and the text of its item in the Edit menu is changed.

doAdjustMenus

If the front window is the Clipboard window, the Close item is enabled, otherwise it is disabled. If the document contains a picture and that picture is currently selected, the Cut, Copy, and Clear items are enabled, otherwise they are disabled.

If the scrap contains data of flavor type `'PICT'` or flavor type 'TEXT', and the front window is not the Clipboard window, the Paste item is enabled, otherwise it is disabled. In this section, `GetCurrentScrap` is called to obtain a reference to the current scrap. This reference is then passed in two calls to `GetScrapFlavorFlags`, which determine whether the scrap contains data of the flavor type `'PICT'` and/or flavor type 'TEXT'. If it does, and if the front window is not the Clipboard window, the Paste item is enabled.

doOpenDocumentWindows

`doOpenDocumentWindows` opens the two document windows, creates document structures for each window, attaches the document structures to the windows and initializes the fields of the document structures.

The `textHdl` field of the first window's document structure is assigned a handle to a 'TEXT' resource and the `textHdl` field of the second window's document structure is assigned a handle to a `'PICT'` resource.

doCloseWindow

`doCloseWindow` closes the Clipboard window (the only window that can be closed from within the program).

If the window is the Clipboard window, the window is disposed of, the global variable that contains its reference is set to `NULL`, the global variable which keeps track of whether the window is showing or hidden is set to false, and the text of the Show/Hide Clipboard menu item is set to "Show Clipboard".

doInContent

`doInContent` handles mouse-down events in the content region of a document window. If the window contains text or a picture, and if the mouse-down was inside the text or picture, the text or picture is selected. If the window contains a text or picture, and if the mouse-down was outside the text or picture, the text or picture is deselected.

The first two lines get a reference to the front window and a handle to its document structure. If the front window is the Clipboard window, the function returns immediately.

doCutCopyCommand

`doCutCopyCommand` handles the user's choice of the Cut and Copy items in the Edit menu.

The first two lines get a reference to the front window and a handle to that window's document structure.

If the `selectFlag` field of the document structure contains false (meaning that the text or picture has not been selected), the function returns immediately. (Note that no check is made as to whether the front window is the Clipboard window because the menu adjustment function disables the Cut and Copy items when the Clipboard window is the front window, meaning that this function can never be called when the Clipboard window is in front.)

`ClearCurrentScrap` attempts to clear the current scrap. (This call should always be

made immediately the user chooses Cut or Copy.) If the call is successful, `GetCurrentScrap` then gets a reference to the scrap.

If the selected item is text, `GetHandleSize` gets the size of the text from the window's document structure. (In a real application, code which gets the size of the selection would appear here.) `PutScrapFlavor` copies the "selected" text to the scrap. If the call to `PutScrapFlavor` is not successful, an alert is displayed to advise the user of the error.

If the selected item is a picture, `GetHandleSize` gets the size of the picture from the window's document structure. `PutScrapFlavor` copies the selected picture to the scrap. If the call to `PutScrapFlavor` is not successful, an alert is displayed to advise the user of the error.

If the menu choice was the Cut item, additional action is taken. Preparatory to a call to `EraseRect`, the current graphics port is saved and the front window's port is made the current port. `DisposeHandle` is called to dispose of the text or picture and the document structure's `textHdl` or `pictureHdl` field, and `selectFlag` field, are set to `NULL` and false respectively. `EraseRect` then erases the port rectangle. (In a real application, the action taken in this block would be to remove the selected text or picture from the document.)

Finally, if the Clipboard window has previously been opened by the user, `doDrawClipboardWindow` is called to draw the current contents of the scrap in the Clipboard window.

doPasteCommand

`doPasteCommand` handles the user's choice of the Paste item from the Edit menu. Note that no check is made as to whether the front window is the Clipboard window because the menu adjustment function disables the Paste item when the Clipboard window is the front window, meaning that this function can never be called when the Clipboard window is in front.

`GetCurrentScrap` gets a reference to the scrap.

The first call to `GetScrapFlavorFlags` determines whether the scrap contains data of flavor type 'TEXT'. If so, `GetScrapFlavorSize` is called to get the size of the 'TEXT' data. `NewHandle` creates a relocatable block of a size equivalent to the 'TEXT' data. `GetScrapFlavorData` is called to copy the 'TEXT' data in the scrap to this block.

`TETextBox` or `DrawThemeTextBox` is called to draw the text in a rectangle equal to the port rectangle minus 40 pixels all round. If the `textHdl` field of the window's document structure does not contain `NULL`, the associated block is disposed of, following which the handle to the block containing the new 'TEXT' data is then assigned to the

`textHdl` field. (In a real application, this block would copy the text into the document at the insertion point. The last three lines in this section simply ensure that, if the window's "document" contains text, it cannot also contain a picture. This is to prevent a picture overdrawing the text when the window contents are updated.)

If the scrap does not contain data of flavor type 'TEXT', `GetScrapFlavorFlags` is called again to determine whether the scrap contains data of flavor type `'PICT'`. If it does, much the same procedure is followed, except that rectangle in which the picture is drawn is extracted from the `'PICT'` data itself and adjusted to the middle of the window via a call to the function `doSetDestRec`.

It is emphasized that the scrap is only checked for flavor type `'PICT'` if the scrap does not contain flavor type 'TEXT'. Thus, if both flavors exist in the scrap, only the 'TEXT' flavor will be used to draw the Clipboard.

doClearCommand

`doClearCommand` handles the user's choice of the Clear item in the Edit menu.

Note that no check is made as to whether the front window is the Clipboard window because the menu adjustment function disables the Clear item when the Clipboard window is the front window.

If the front window's document structure indicates that the window contains text or a picture, the block containing the TextEdit structure or Picture structure is disposed of and the relevant field of the document structure is set to `NULL`. In addition, the `selectFlag` field of the document structure is set to false and the window's port rectangle is erased.

doClipboardCommand

`doClipboardCommand` handles the user's choice of the Show/Hide Clipboard item in the Edit menu.

The first line gets a reference to the Edit menu. This will be required in order to toggle the Show/Hide Clipboard item's text between Show Clipboard and Hide Clipboard.

The if statement checks whether the Clipboard window has been created. If not, the Clipboard window is created by the call to `GetNewCWindow`, a document structure is created and attached to the window, the `windowType` field of the document structure is set to indicate that the window is of the Clipboard type, the Show/Hide Clipboard menu item text is set, the window's special window event handler is installed, the window is shown, and a global variable which keeps track of whether the Clipboard window is currently showing or hidden is set to true.

If the Clipboard window has previously been created, and if the window is currently showing, the window is hidden, the Clipboard-showing flag is set to false, and the Show/Hide Clipboard item's text is set to "Show Clipboard". If the window is not currently showing, the window is made visible, the Clipboard-showing flag is set to true, and the Show/Hide Clipboard item's text is set to "Hide Clipboard".

doDrawClipboardWindow

`doDrawClipboardWindow` draws the contents of the scrap in the Clipboard window. It supports the drawing of both `'PICT'` and 'TEXT' flavor type data.

The first four lines save the current graphics port, make the Clipboard window's graphics port the current graphics port and erase the window's content region.

`DrawThemeWindowHeader` draws a window header in the top of the window. Text describing the type of data in the scrap will be drawn in this header.

The text mode for text drawing is set at the next four lines, following which "Clipboard Contents:" is drawn in the header.

The code for getting a reference to the current scrap, checking for the 'TEXT' and `'PICT'` flavor types, getting the flavor size, getting the flavor data, and drawing the text and picture in the window is much the same as in the function `doPasteCommand`. The differences are: the rectangle in which the text is drawn is the port rectangle minus two pixels all round and with the top further adjusted downwards by the height of the window header; the left/top of the rectangle in which the picture is drawn is two pixels inside the left side of the content region and two pixels below the window header respectively; the words "Text" or "Picture" are drawn in the window header as appropriate.

Note that, as was the case in the function doPasteCommand, the scrap is only checked for flavor type `'PICT'` if the scrap does not contain flavor type 'TEXT'. Thus, if both flavors exist in the scrap, only the 'TEXT' flavor will be used to draw the Clipboard.

doDrawDocumentWindow

`doDrawDocumentWindow` draws the text or picture belonging to that window in the window. It is called when the `kEventWindowDrawContent` event type is received for the window.

Chapter 21. Text, Textedit, Dates, Times, and Numbers

IN THIS CHAPTER

- Introduction
- More on Text
- Introduction to TextEdit
- Monostyled TextEdit
- Multistyled TextEdit
- Formatting and Displaying Dates, Times, and Numbers
- Main TextEdit Constants, Data Types and Functions
- Main Constants, Data Types and Functions Relating to Dates, Times and Numbers
- Demonstration Program MonoTextEdit Listing
- Demonstration Program MonoTextEdit Comments
- Demonstration Program DateTimeNumbers Listing
- Demonstration Program DateTimeNumbers Comments

Introduction

The subject of text on the Macintosh is a complex matter involving QuickDraw, TextEdit, the Font Manager, the Text Utilities, the Script Manager, the Text Services Manager, Apple Type Services for Unicode Imaging, the Resource Manager, keyboard resources, and international resources. Part of that complexity arises from the fact that the system software supports many different writing systems, including Roman, Chinese, Japanese, Hebrew, and Arabic.

> **Note**
>
> Some of the information in this chapter is valid only in the case of the Roman writing system.

Text on the Macintosh was touched on briefly in Chapter 12, which included descriptions of QuickDraw functions used for drawing text and for setting the font, style, size, and transfer mode. Chapter 15 contained a brief treatment of considerations applying to the printing of text. Chapter 26 addresses the Multilingual Text Engine (MLTE) introduced with Mac OS 9.

This chapter addresses

- TextEdit, which you can use to provide your application with basic text editing and formatting capabilities.

  Note that the emphasis in this chapter is on monostyled TextEdit. With Mac OS 9 and the introduction of the Multilingual Text Engine (see Chapter 26), it became all but inconceivable that programmers would ever again use multistyled TextEdit to provide their applications with multistyled text-editing capabilities. In the following, multistyled TextEdit is addressed only to the extent necessary to support an understanding of the display of noneditable multistyled text, as in the Help dialog component of the demonstration program MonoTextEdit associated with this chapter.

- The formatting and display of dates, times, and numbers.

Before addressing those particular subjects, however, a brief overview of various closely related matters is appropriate.

More on Text

Characters and Character Sets and Codes

A **character** is a symbol that represents the concept of, for example, a lowercase b, the number 2, or the arithmetic operator +, for example. A collection of characters is called a **character set.** Individual characters within a character set are identified by a **character code.**

The **Apple Standard Roman character set** is the fundamental character set for the Macintosh computer. As shown in Fig 1, it uses all character codes from `0x00` to `0xFF`. The Standard Roman character set is actually an extended version of the **ASCII character set,** which uses character codes from `0x00` to `0x7F` only, and which is highlighted in Fig 1.

Figure 1. ASCII CHARACTER SET

	LOW ASCII RANGE								*HIGH ASCII RANGE*							
	CONTROL CODES		*ROMAN CHARACTERS*						*SCRIPT-SPECIFIC CHARACTERS*							
	0x	*1x*	*2x*	*3x*	*4x*	*5x*	*6x*	*7x*	*8x*	*9x*	*Ax*	*Bx*	*Cx*	*Dx*	*Ex*	*Fx*
x0	nul	dle	sp	0	@	P	`	p	Ä	ê	†	∞	¿	–	‡	?
x1	soh	DC1	!	1	A	Q	a	q	Å	ë	°	±	¡	—	·	Ò
x2	stx	DC2	"	2	B	R	b	r	Ç	í	¢	≤	¬	"	‚	Ú
x3	etx	DC3	#	3	C	S	c	s	É	ì	£	≤	√	"	„	Û
x4	eot	DC4	$	4	D	T	d	t	Ñ	î	§	¥	ƒ	'	‰	Ù
x5	enq	nak	%	5	E	U	e	u	Ö	ï	•	µ	≈	'	Â	ı
x6	ack	syn	&	6	F	V	f	v	Ü	ñ	¶	∂	Δ	÷	Ê	^
x7	bel	etb	'	7	G	W	g	w	á	ó	ß	∑	«	◊	Á	~
x8	bs	can	(	8	H	X	h	x	à	ò	®	∏	»	ÿ	Ë	¯
x9	ht	em	)	9	I	Y	i	y	â	ô	©	π	…	Ÿ	È	˘
xA	lf	sub	*	:	J	Z	j	z	ä	ö	™	∫		/	Í	˙
xB	vt	esc	+	;	K	[	k	{	ã	õ	´	ª	À		Î	°
xC	ff	fs	,	<	L	\	l	|	å	ú	¨	º	Ã	‹	Ï	¸
xD	cr	gs	-	=	M	]	m	}	ç	ù	≠	Ω	Õ	›	Ì	˝
xE	so	rs	.	>	N	^	n	~	é	û	Æ	æ	Œ	fi	Ó	˛
xF	si	us	/	?	O	_	o	del	è	ü	Ø	ø	œ	fl	Ô	ˇ

Glyphs

The visual representation of a character on a display device is called a **glyph.** In other words, a glyph is the shape by which a character is represented. A specific character can be represented by many different shapes (that is, glyphs).

Two types of glyphs are used by the Font Manager: **bitmapped glyphs** and glyphs from **outline fonts.** A bitmapped glyph is a bitmap designed at a fixed size for a particular display device. An "outline" is a mathematical description of the glyph in terms of lines and curves, and is used by the Font Manager to create bitmaps at any size for any display device.

Typefaces

If all glyphs for a character set share certain design characteristics, they form a **typeface.** Typefaces have their own names, such as Arial, Geneva, or Times.

Styles

A specific variation in a glyphs appearance is called a **style.** On the Macintosh, available styles include plain, bold, italic, underline, outline, shadow, condensed, and extended. QuickDraw can add styles to bitmaps, or fonts can be designed with a specific style, such as Arial Italic.

Fonts and Font Families

A **font** is a full set of glyphs in a specific typeface and style. All fonts have a font name, such as Arial or Geneva, which is ordinarily the same name as the typeface from which it was derived. Except for fonts not in the plain style, the font's name includes the style (or styles), for example "Palatino Bold Italic."

Fonts on the Macintosh are resources. The resource types are as follows:

- Bitmapped font resources are of type `'FONT'` (the original resource type for fonts) and `'NFNT'` (bitmapped font). `'FONT'` and `'NFNT'` resources provide a separate bitmap for each glyph in each style and size.
- Outline font resources are of type `'sfnt'`. `'sfnt'` resources comprise glyphs in a particular typeface and style.

If multiple fonts of the same typeface are present, the Font Manager groups those fonts into **font families** of resource type `'FOND'`. A **font family ID** is the resource ID for a font family.

As an aside, most (though not all) fonts assign glyphs to character codes `0x20` to `0x7F`, which visually define the characters associated with those codes. However, there are differences in the glyphs assigned to the high-ASCII range. Indeed, some fonts do not actually include glyphs for

all, or part, of the high-ASCII range.

Note

Fonts such as Zapf Dingbats assign glyphs of pictorial symbols to this range. as well as the low-ASCII range.

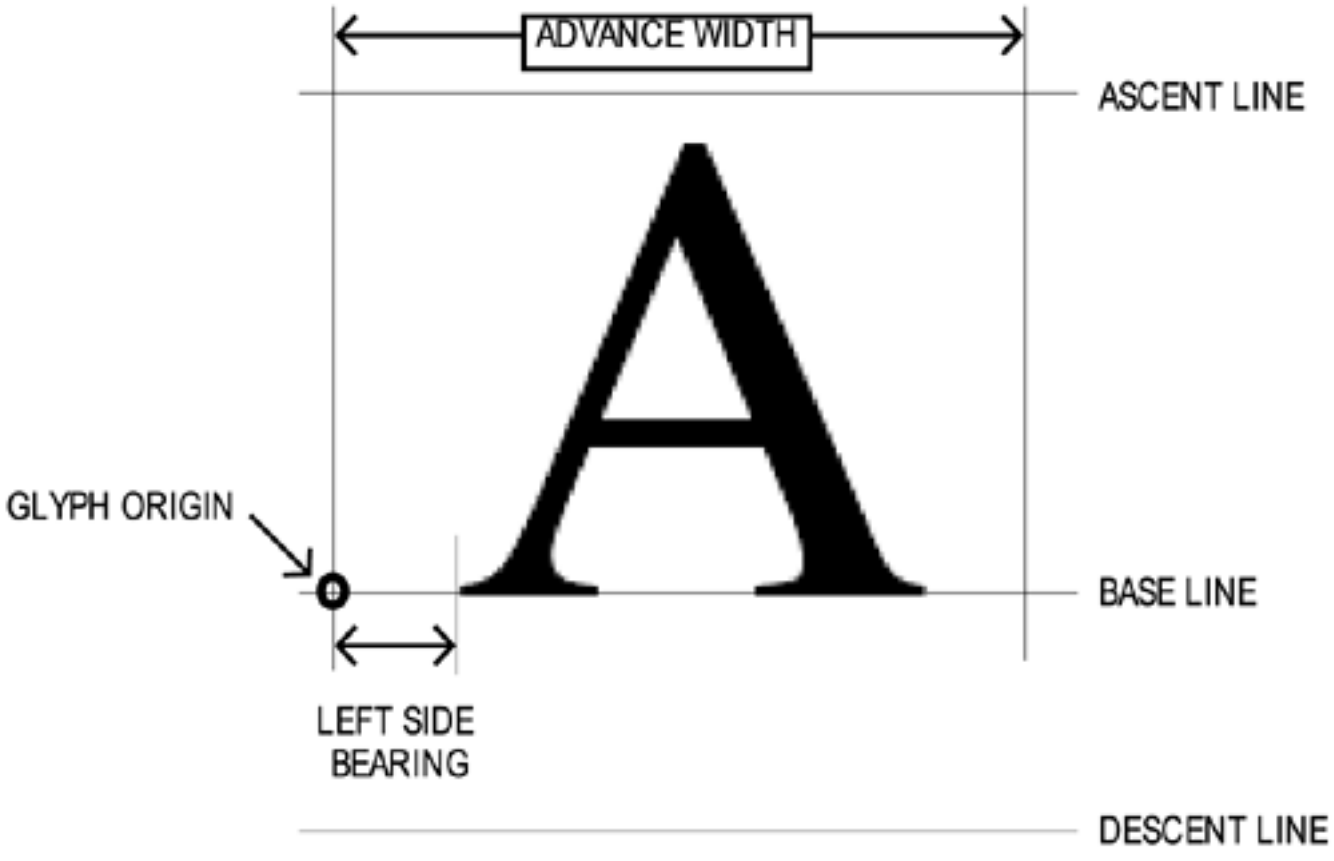
Font Measurements

Fonts are either **monospaced** or **proportional.** All glyphs in a monospaced font are the same width. The glyphs in a proportional font have different widths, m being wider than i, for example.

Base Line, Ascent Line, and Descent Line

Most glyphs in a font sit on an imaginary line called the **base line.** The **ascent line** approximately corresponds with the tops of the uppercase letters of the font. The **descent line** usually corresponds to the bottom of descenders (the tails of glyphs like "j"). See Fig 2.

Figure 2. FONT MEASUREMENT TERMS



Glyph Origin, Left Side Bearing, and Advance Width

The **glyph origin** is where QuickDraw begins drawing the glyph. The **left side bearing** is the white space between the glyph origin and the beginning of the glyph. The **advance width** is the full width of a glyph, measured from its origin to the origin of the next glyph. See Fig 2.

Font Size

Font size is the measurement, in **points,** from the base line of one line of text to the base line of the next line (assuming single-spaced text). A point is equivalent to 1/72 of an inch. The size of a font is often, but not always, the sum of the ascent, descent, and **leading** (pronounced "ledding") values for a font. (The leading is the vertical space between the descent line of one line of single-spaced text and the ascent line of the next line.)

The Font Manager and QuickDraw

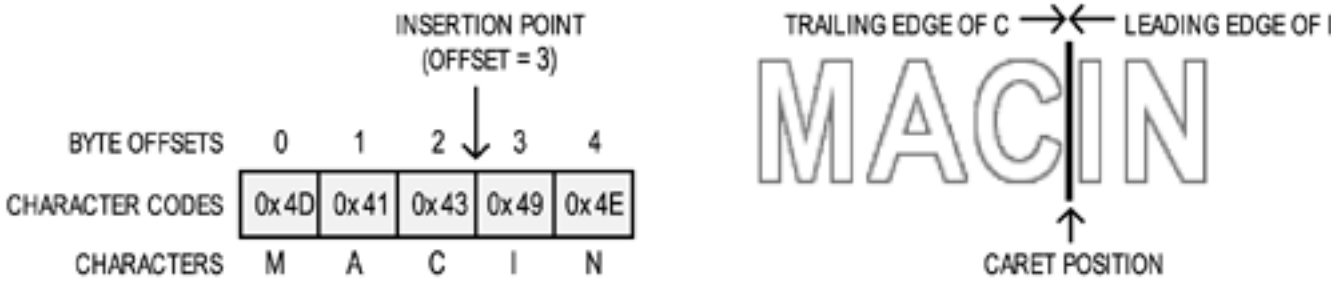
The Font Manager keeps track of all fonts available to an application and supports QuickDraw by providing the character bitmaps that QuickDraw needs. If QuickDraw requests a typeface that is not represented in the available fonts, the Font Manager substitutes one that is. Where necessary, QuickDraw scales the font to the requested size and applies the specified style.

Caret Position and Text Offset

In the world of text editing, the **caret** is defined as the blinking vertical bar that indicates the **insertion point** in text, and a **caret position** is a location on the screen that corresponds to an insertion point in memory. A caret position is always *between* glyphs on the screen. The caret is always positioned on the leading edge of the glyph corresponding to the character at the insertion point in memory. When a new character is inserted, the character at the insertion point, and all subsequent characters, are shifted forward one character position in memory.

The relationship between caret position, insertion point and offset is illustrated in Fig 3.

Figure 3. CARET POSITION AND INSERTION POINT

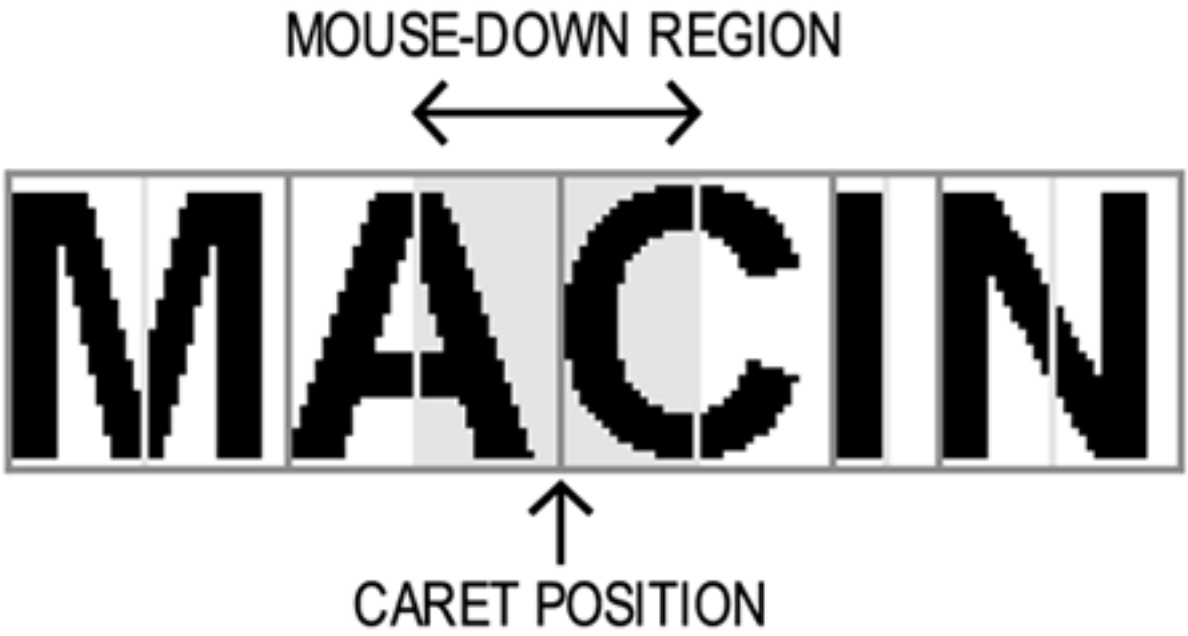


Converting Screen Position to Text Offset

A mouse-down event can occur anywhere in a glyph; however, the caret position derived from that mouse-down must be an infinitely thin line between two glyphs.

As shown in Fig 4, a line of glyphs is divided into **mouse-down regions,** which, except at the end of the line, extend from the center of one glyph to the center of the next glyph. A click anywhere in a particular mouse-down region yields the same caret position.

Figure 4. INTERPRETING CARET POSITION FROM A MOUSE-DOWN EVENT



Selection Range and Insertion Points

The **selection range** is the sequence of zero or more contiguous characters where the next text editing operation is to occur. If a selection range contains zero characters, it is called an **insertion point.**

Highlighting

A selection range is typically marked by **highlighting**-that is, by drawing the glyphs with a colored background. The limits of highlighting rectangles are measured in terms of caret position. For example, if the characters A, C, and I in Fig 3 were highlighted, the highlighting would extend from the leading edge of A (offset = 1) to the leading edge of N (offset = 4).

Outline Highlighting

Outline highlighting is the "framing" of text in the selection range in an inactive window. If there is no selection range, a gray, unblinking caret is displayed.

Keyboards and Text

Each keypress on a particular keyboard generates a value called a **raw key code.** The keyboard driver which handles the keypress uses the **key-map** (`'KMAP'`) **resource** to map the raw key code to a keyboard-independent **virtual key code.** It then uses the Event Manager and the **keyboard layout** (`'KCHR'`) **resource** to convert a virtual keycode into a character code. The character code is passed to your application in the event structure generated by the keypress.

Introduction to TextEdit

TextEdit is a collection of functions and data structures that you can use for the purposes of basic text formatting and editing. It was originally designed to handle edit text items in dialogs, and was subsequently enhanced to provide some of the more complex capabilities required of a basic text editor. That said, it should be understood that TextEdit was never intended to support all of the basic features generally required of a text editor (for example, tabs) and was never intended to manipulate lengthy text documents in excess of 32 KB. Indeed, the limit for documents created by TextEdit is 32,767 characters.

If you do not need to create large files and only need basic formatting capabilities, TextEdit provides a useful alternative to writing your own specialized text processing functions.

Editing Tasks Performed by TextEdit

The fundamental editing tasks that TextEdit can perform for your application are as follows:

- Selecting text by clicking and dragging the mouse, selecting words by double-clicking, and extending or shortening selections by Shift-clicking.
- Displaying the caret at the insertion point or highlighting the current text selection, as appropriate.
- Handling line breaking, that is, preventing a word from being split between lines.
- Cutting, copying, and pasting text within your application, and between your application and other applications.
- Managing the use of more than one font, text size, text color, and text style from character to character.

TextEdit Options

You can use TextEdit at different levels of complexity.

Using TextEdit Indirectly

For the simplest level of text handling (that is, in dialogs), you need not even call TextEdit directly but rather use the Dialog Manager. The Dialog Manager, in turn, calls TextEdit to edit and display text.

Displaying Static Text

If you simply want to display one or more lines of static (non-editable) text, you can call `TETextBox`, which draws your text in the location you specify. `TETextBox` may be used to display text that you cannot edit. You do not need to create a **TextEdit structure** because `TETextBox` creates its own TextEdit structure. `TETextBox` draws the text in a rectangle whose size you specify in the coordinates of the current graphics port. Using the following constants, you can specify how text is aligned in the box:

Constant	*Description*
`teFlushDefault`	Default alignment according to primary line direction of the script system. (Left for Roman script system.)
`teCenter`	Center alignment.
`teFlushRight`	Right alignment.
`teFlushLeft`	Left alignment.

Text Handling-Monostyled Text

If your application requires very basic text handling in a single typeface, style, and size, you probably only need **monostyled TextEdit.** You can use monostyled TextEdit with any single available font.

Text Handling-Multistyled Text

If your application requires a somewhat higher level of text handling (allowing the user to change typeface, style, and size within the document, for example), you can use **multistyled TextEdit.** However, as previously stated, mutistyled TextEdit has now been overshadowed by the introduction of the Multilingual Text Engine.

Caret Position and Movement in TextEdit

TextEdit always positions the caret where the next editing operation will occur. When TextEdit pastes text, it positions the caret after the newly pasted text. Assuming that the caret is not in the first or last line of text, TextEdit moves the caret up or down one line when the user presses the Up Arrow key or the Down Arrow key. (If the caret is on the first line, TextEdit moves the caret to the beginning of text on that line if the user presses the Up Arrow key. If the caret is on the last line, TextEdit moves the caret to the end of the text on that line if the user presses the Down Arrow key.)

> **Note**
>
> TextEdit does not support the use of modifier keys, such as the Shift key, in conjunction with the arrow keys.

Automatic Scrolling

One way for the user is to select large blocks of text is to click in the text and, holding the mouse button down, drag the cursor above, below, left of, or right of TextEdit's **view rectangle.** While the mouse button remains down, and provided that your application has enabled automatic scrolling, TextEdit continually calls its **click loop function** to automatically scroll the text.

Although TextEdit's default click loop function automatically scrolls the text, it cannot adjust the scroll box/scroller position in an application's scroll bars to follow up the scrolling. The default click loop function can, however, be replaced with an application-defined click loop (callback) function which accommodates scroll bars.

TextEdit's Private, Null, and Style Scraps

Internally, TextEdit uses three scrap areas, namely, the **private scrap,** the **null scrap,** and the **style scrap.** The null scrap and the style scrap apply only to multistyled TextEdit.

The private scrap, which belongs to your application, is used for all cut, copy, and paste activity.

The null scrap is used by TextEdit to store **character attribute** information associated with a null selection or text that is deleted by backspacing. (A null selection is an insertion point.)

When multistyled text is cut or copied, TextEdit copies character attribute information to the style scrap.

> **Note**
>
> The font, style, size, and color aspects of text are collectively referred to as **character attributes.**

Text Alignment

Text **alignment** can be left-aligned, right-aligned, centered, or justified. Justified means aligned with both the left and right edges of TextEdit's **destination rectangle,** and is achieved by spreading or compressing text to fit a given line width.

Primary TextEdit Data Structures

The primary data structures used by TextEdit are the TextEdit structure and the **dispatch structure.** Additional data structures are associated with multistyled TextEdit. This section describes the primary data structures only.

The TextEdit Structure

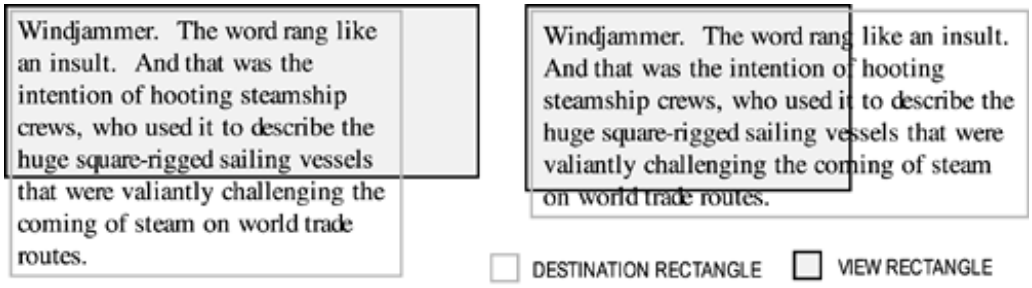
The TextEdit structure is the principal data structure used by TextEdit. This structure is the same regardless of whether the text is monostyled or multistyled, although some fields are used differently for multistyled TextEdit structures. The TextEdit structure is as follows:

```
struct TERec
{
  Rect           destRect;      // Destination rectangle.
  Rect           viewRect;      // View rectangle.
  Rect           selRect;       // Selection rectangle.
  short          lineHeight;    // Vert spacing of lines. -1 in multistyled.
  short          fontAscent;    // Font ascent. -1 in multistyled TextEdit structure.
  Point          selPoint;      // Point selected with the mouse.
  short          selStart;      // Start of selection range.
  short          selEnd;        // End of selection range.
  short          active;        // Set when structure is activated or deactivated.
  WordBreakUPP   wordBreak;     // Word break function.
  TEClickLoopUPP clickLoop;     // Click loop function.
  long           clickTime;     // (Used internally.)
  short          clickLoc;      // (Used internally.)
  long           caretTime;     // (Used internally.)
  short          caretState;    // (Used internally.)
  short          just;          // Text alignment.
  short          teLength;      // Length of text.
  Handle         hText;         // Handle to text to be edited.
  long           hDispatchRec;  // Handle to TextEdit dispatch structure.
  short          clikStuff;     // (Used internally)
  short          crOnly;        // If < 0, new line at Return only.
  short          txFont;     // Text font.  // If multistyled edit struct (txSize = -1),
  StyleField     txFace;     // Chara style. // these bytes are used as a handle
  SInt8          filler;     //              // to a style structure (TEStyleHandle).
  short          txMode;        // Pen mode.
  short          txSize;        // Font size. -1 in multistyled TextEdit structure.
  GrafPtr        inPort;        // Pointer to grafPort for this TextEdit structure.
  HighHookUPP    highHook;      // Used for text highlighting, caret appearance.
  CaretHookUPP   caretHook;     // Used from assembly language.
  short          nLines;        // Number of lines.
  short          lineStarts[16001];  // Positions of line starts.
} ;
typedef struct TERec TERec;
typedef TERec *TEPtr;
typedef TEPtr *TEHandle;
```

Field Descriptions

`destRect` The destination rectangle (local coordinates), which is the area in which text is drawn (see Fig 5). The top of this rectangle determines the position of the first line of text and the two sides determine the beginning and the end of each line. The bottom of the rectangle varies as text is added or removed as a result of editing operations.

Figure 5. DESTINATION AND VIEW RECTANGLES


Windjammer. The word rang like an insult. And that was the intention of hooting steamship crews, who used it to describe the huge square-rigged sailing vessels that were valiantly challenging the coming of steam on world trade routes.
Windjammer. The word rang like an insult. And that was the intention of hooting steamship crews, who used it to describe the huge square-rigged sailing vessels that were valiantly challenging the coming of steam on world trade routes.
DESTINATION RECTANGLE
VIEW RECTANGLE

The destination rectangle is central to the matter of scrolling text. When text is scrolled downward; for example, you can think of the destination rectangle as being moved upwards through the view rectangle.

`viewRect` The view rectangle (local coordinates), which is the area in which text is actually displayed (see Fig 5).

`selRect` The selection rectangle boundaries (local coordinates).

`lineHeight` In a monostyled TextEdit structure, the vertical spacing of lines of text, that is, the distance from the ascent line of any one line of text to the ascent line of the next line of text. In a multistyled TextEdit structure, this field is set to -1, which indicates that line heights are calculated for each individual line of text.

`fontAscent` In a monostyled TextEdit structure, the font ascent, that is, the vertical distance above the baseline the pen is positioned to begin drawing the caret or selection highlighting. (In the case of single-spaced text, the font ascent is the height of the text in pixels.) In a multistyled TextEdit structure, this field is set to -1, which indicates that font ascent is calculated for each individual line of text.

`selPoint` The point selected with the mouse (local coordinates).

`selStart` The byte offset of the start of the selection range. TextEdit initializes this field to 0 when you create a TextEdit structure.

`selEnd` The byte offset of the end of the selection range. TextEdit initializes this field to 0 when you create a TextEdit structure. With both `selStart` and `selEnd` initialized to 0, the insertion point is placed at the beginning of the text.

`active` Set when the TextEdit structure is activated and reset when the TextEdit structure is rendered inactive.

`wordBreak` Universal procedure pointer to the word selection break function, which determines the word that is highlighted when the user double-clicks in the text and the position at which text is wrapped at the end of the line.

`clickLoop` Universal procedure pointer to the click loop function, which is called repeatedly while the mouse button is held down within the text.

`just` Text alignment (default, left, center, or right).

`teLength` The number of bytes in the text. The maximum allowable length is 32,767 bytes. When you create a TextEdit structure, TextEdit initializes this field to 0.

`hText` A handle to the text. When you create a TextEdit structure, TextEdit initializes this field to point to a zero-length block in the application heap.

`hDispatchRec` The handle to the TextEdit dispatch structure (see below). For internal use only.

`clikStuff` TextEdit sets this field according to whether the most recent mouse-down event occurred on a glyph's leading or trailing edge. Used internally by TextEdit to determine a caret position.

`crOnly` If the value in this field is positive, text wraps at the right edge of the destination rectangle. If the value is negative, text does *not* wrap.

`txFont` In a monostyled TextEdit structure, the font of all the text in the TextEdit structure. (If you change the value, you should

also change the `lineHeight` and `fontAscent` fields as appropriate.) In a multistyled TextEdit structure, if the `txSize` field (see below) is set to -1, this field combines with `txFace` and `filler` to hold a handle to the associated style structure.

`txFace` In a monostyled TextEdit structure, the character attributes of all the text in a TextEdit structure. (If you change this value, you should also change the `lineHeight` and `fontAscent` fields as appropriate.) In a multistyled TextEdit structure, if the `txSize` field is set to -1, this field combines with `txFont` and `filler` to hold a handle to the associated style structure.

`txMode` The pen mode of all the text.

`txSize` In a monostyled TextEdit structure, this field is set to the size of the text in points. In a multistyled TextEdit structure, this field is set to is -1, indicating that the TextEdit structure contains associated character attribute information. The `txFont`, `txFace`, and `filler` fields combine to form a handle to the style structure in which this character attribute information is stored.

`inPort` A pointer to the graphics port associated with the TextEdit structure.

`highHook` A universal procedure pointer to the function that deals with text highlighting.

`caretHook` Universal procedure pointer to the function that controls the appearance of the caret.

`numLines` The number of lines of text.

`lineStarts` A dynamic array that contains the character position of the first character in each line of the text. This array grows and shrinks, containing only as many elements as needed.

The Dispatch Structure

The `hDispatchRec` field of the TextEdit structure stores a handle to the dispatch structure. The dispatch structure is an internal data structure whose fields contain the addresses of functions that determine the way TextEdit behaves. You can modify TextEdit's default behavior by replacing the address of a default function in the dispatch structure with the address of your own customized function.

Monostyled TextEdit

This section describes the use of TextEdit with monostyled text, that is, text with a single typeface, style, and size. Everything in this section also applies to using TextEdit with multistyled text except where otherwise indicated.

Creating, and Disposing of, a Monostyled TextEdit Structure

Creating a Monostyled TextEdit Structure

To use TextEdit functions, you must first create a TextEdit structure using `TENew`. `TENew` returns a handle to the newly-created monostyled TextEdit structure. You typically store the returned handle in a field of a document structure, the handle to which is typically stored in the application window's `refCon` field.

The required destination and view rectangles are specified in the `TENew` call. You should inset the destination rectangle at least four pixels from the left and right edges of the graphics port, making an additional allowance for scroll bars as appropriate. This will ensure that the first and last glyphs in each line are fully visible. You typically make the view rectangle equal to the destination rectangle. (If you do not want the text to be visible, specify a view rectangle off the screen.)

When a TextEdit structure is created, TextEdit initializes the TextEdit structure's fields based on values in the current graphics port object and on the type of TextEdit structure you create.

Disposing of a TextEdit Structure

Memory allocated for a TextEdit structure may be released by calling `TEDispose`.

Setting the Text

A new TextEdit structure does not contain any text until the user either opens an existing document or enters text via the keyboard. The following is concerned with existing documents.

`TESetText` may be used to specify the text to be edited. Alternatively, you can set the `hText` field of the TextEdit structure directly.

Calling TESetText

When a user opens a document, your application can load that document's text and then call `TESetText`. `TESetText` creates a copy of the text and stores the copy in the existing handle of the TextEdit structure's `hText` field.

You must pass the length of the text in the call to `TESetText`. `TESetText` uses this to reset the `teLength` field of the TextEdit structure, and to set the `selStart` and `selEnd` fields to the last byte offset of the text. `TESetText` also calculates the line breaks.

`TESetText` does not cause the text to be displayed immediately. You must call `InvalWindowRect` to force the text to be displayed at the next update event for the active window.

Changing the hText Field

The alternative of setting the `hText` field directly, replacing the existing handle with the handle of the new text, saves memory if you have a lot of text. When you use this method, you must also assign the length of the text to the `teLength` field of the TextEdit structure and call `TECalText` to recalculate the `lineStarts` array and `numLines` values.

Responding to Events

Activate Events

When your application receives an activate event (Classic event model) or `kEventWindowActivated` or `kEventWindowDeactivated` event type (Carbon event model), it should call `TEActivate` or `TEDeactivate` as appropriate.

A TextEdit structure which has been activated by `TEActivate` has its selection highlighted or, if there is no selection, has its caret displayed and blinking at the insertion point. A TextEdit structure which has been deactivated by `TEDeactivate` has its selection range outlined (if outline highlighting is enabled) or, if there is no selection, has a gray, unblinking caret displayed at the insertion point.

> **Note**
>
> Outline highlighting may be activated and deactivated using `TEFeatureFlag`.

Note that when you use `TEClick` and `TESetSelect` to set the selection range or insertion point, the selection range is not highlighted, or the blinking caret is not displayed,

until the TextEdit structure is activated. (However, if outline highlighting is enabled, the text of the selection range will be framed or a gray, unblinking caret will be displayed.)

Update Events-Calling TEUpdate

When your application receives an update event (Classic event model) or `kEventWindowDrawContent` or `kEventWindowUpdate` event type (Carbon event model), it should call `TEUpdate`. In addition, you should call `TEUpdate` after changing any fields of the TextEdit structure, or after any editing or scrolling operation, which alters the onscreen appearance of the text.

Mouse-Down Events-Calling TEClick

On receipt of a mouse-down event that should be handled by TextEdit, your application must pass the event to `TEClick`. `TEClick` tells TextEdit that a mouse-down event has occurred. Before calling `TEClick`, however, your application must

- Convert the mouse location from global coordinates to the local coordinates required by `TEClick`.
- Determine if the Shift key was down at the time of the event.

`TEClick` repeatedly calls the click loop function as long as the mouse button is held down and retains control until the button is released. The behavior of `TEClick` depends on whether the Shift key was down at the time of the mouse-down event and on other user actions as follows:

User's Action	***Behavior of TEClick***
Shift key down.	Extend the current selection range.
Shift key not down.	Remove highlighting from current selection range. Position the insertion point as close as possible to the location of the mouse click.
Mouse dragged.	Expand or shorten the selection range a character at a time. Keep control until the user releases the mouse button.
Double-click.	Extend the selection to include the entire word where the cursor is positioned.

Key-Down Events-Accepting Text Input

On receipt of a key-down event that should be handled by TextEdit, your application must call `TEKey` to accept the keyboard input. `TEKey` replaces the current selection range with

the character passed to it and moves the insertion point just past the inserted character.

Depending on the requirements of your application, you may need to filter out certain character codes (for example, that for a Tab key press) so that they are not passed to `TEKey`. You should also check that the TextEdit limit of 32,767 bytes will not be exceeded by the insertion of the character before calling `TEKey` and you should call your scroll bar adjustment function immediately after the insertion.

Caret Blinking

To force the insertion point caret to blink, your application must call `TEIdle` at an interval equal to the value stored in the low-memory global `CaretTime`. You can retrieve this value by calling `GetCaretTime`. In Classic event model applications, you should set the `sleep` parameter in the `WaitNextEvent` call to this value and call `TEIdle` when `WaitNextEvent` returns `0` with a null event. In Carbon event model applications, you should install a timer set to fire at this interval and call `TEIdle` when the timer fires.

If there is more than one TextEdit structure associated with an active window, you must ensure that you pass `TEIdle` the handle to the currently active TextEdit structure. You should also check that the handle to be passed to `TEIdle` does not contain `NULL` before calling the function.

Cutting, Copying, and Pasting

You can use TextEdit to cut, copy, and paste text within and between TextEdit structures, and across applications. The relevant functions, and their effect in the case of a monostyled TextEdit structure, are as follows:

Function	*Use To*	*Comments*
`TECut`	Cut text.	Copies the text to the TextEdit private scrap.
`TECopy`	Copy text.	Copies the text to the TextEdit private scrap.
`TEPaste`	Paste text.	Pastes from the TextEdit private scrap to the TextEdit structure.
`TEToScrap`	Copy TextEdit private scrap to the Carbon Scrap Manager's scrap.	Copying via the Carbon Scrap Manager's scrap is required if monostyled text is to be carried across applications.

`TEFromScrap`	Copy the Carbon Scrap Manager's scrap to TextEdit private scrap.	Copying via the Carbon Scrap Manager's scrap is required if text is to be carried across applications.
`TEGetScrapLength`	Determine the length of the text to be pasted.	Returns the size, in bytes, of the text in the private scrap.

You will need to call your vertical scroll bar adjustment function immediately after cut and paste operations. In addition, you will need to ensure that a paste will not cause the TextEdit limit of 32,767 bytes to be exceeded.

Inserting and Deleting Text

The following TextEdit functions are used to insert and delete monostyled text:

Function	*Use To*	*Comments*
`TEInsert`	Insert text into the TextEdit structure immediately before the selection range or insertion point.	Does not affect the selection range. Redraws the text if necessary.
`TEDelete`	Remove the selected range of text from the TextEdit structure.	Does not transfer the text to either TextEdit's private scrap or the Carbon Scrap Manager's scrap. Useful for implementing a Clear command. Redraws the remaining text if necessary.

You will need to call your vertical scroll bar adjustment function immediately after insertions and deletions. In addition, you will need to ensure that an insertion will not cause the TextEdit limit of 32,767 bytes to be exceeded.

Setting the Selection Range or Insertion Point

Using the `TESetSelect` function, your application can set the selection range or set the location of the insertion point. (For example, your application might use `TESetSelect` to locate the caret at the start of a data entry field where you want the user to enter a value.)

`TESetSelect` changes the value in the `selStart` and `selEnd` fields of the TextEdit structure.

To set a selection range, you pass the byte offsets of the starting and ending characters in the `selStart` and `selEnd` parameters. To set the location of the insertion point, you pass the same values in the `selStart` and `selEnd` parameters. You can set the selection range (or insertion point) to any character position corresponding to byte offsets 0 to 32767.

To implement a Select All menu command, pass 0 in the `selStart` parameter and the value in the `teLength` field of the TextEdit structure in the `selEnd` parameter.

Enabling and Disabling

You can use the `TEAutoView` function to enable automatic scrolling (which, by default, is disabled). `TEAutoView` may also be used to disable automatic scrolling.

Customizing

As previously stated, the default click loop (callback) function does not adjust the scroll bars as the text is scrolled, a situation that can be overcome by replacing the default click loop function with an application-defined click loop (callback) function which updates the scroll bars as it scrolls the text.

The `clickLoop` field of the TextEdit structure contains a universal procedure pointer to a click loop (callback) function, which is called continuously as long as the mouse button is held down. Installing your custom function involves a call to `TESetClickLoop` to assign the universal procedure pointer to the TextEdit structure's `clickLoop` field.

Scrolling Text

When a mouse-down event occurs in a scroll bar, your application must determine how far to scroll the text. The basic value for vertical scrolling of monostyled text is typically the value in the `lineHeight` field of the TextEdit structure, which can be used as the number of pixels to scroll for clicks in the Up and Down scroll arrows. For clicks in the gray areas/track, this value is typically multiplied by the number of text lines in the view rectangle minus 1. Scrolling by dragging the scroll box/scroller involves determining the number of text lines to scroll based on the current position of the top of the destination rectangle and the control value on mouse button release.

You pass the number of pixels to scroll in a call to `TEScroll` or `TEPinScroll`. (The difference between these two functions is that the latter stops scrolling when the last line is scrolled into the view rectangle.) The destination rectangle is offset by the amount you scroll.

Forcing the Selection Range into the View

Your application can call `TESelView` to force the selection range to be displayed in the view rectangle. When automatic scrolling is enabled, `TESelView` scrolls the selection range into view, if necessary.

Setting Text Alignment

You can change the alignment of the entire text of a TextEdit structure by calling `TESetAlignment`. The following constants apply:

Constant	*Description*
`teFlushDefault`	Default alignment according to primary line direction of the script system. (Left for Roman script system.)
`teCenter`	Center alignment.
`teFlushRight`	Right alignment.
`teFlushLeft`	Left alignment.

You should call the Window manager's `InvalWindowRect` function after you change the alignment so that the text is redrawn in the new alignment.

Saving and Opening TextEdit Documents

The demonstration program in Chapter 18 demonstrates opening and saving monostyled TextEdit documents.

Multistyled TextEdit

With the introduction in Mac OS 9 of the Multilingual Text Editor (see Chapter 26), it became all but inconceivable that programmers would ever again use multistyled TextEdit to provide their applications with multistyled text editing capabilities. That said, multistyled TextEdit may still be considered useful where the requirement is simply the display of noneditable multistyled text, as in the Help dialog component of the demonstration program associated with this chapter.

This section addresses additional factors and considerations applying to multistyled TextEdit, but only to the extent necessary to support an understanding of those factors involved in the display of noneditable styled text, as in the Help dialog component of the demonstration program associated with this chapter.

Text With Multiple Styles-Style Runs, Text Segments, Font Runs, Character Attributes

Text that uses a variety of fonts, styles, sizes, and colors is referred to as **multistyled text.**

TextEdit organizes multistyled text into **style runs,** which comprise a set of contiguous characters that all share the same font, size, style, and color characteristics. TextEdit tracks style runs in the data structures allocated for a multistyled TextEdit structure and uses this information to correctly display multistyled text.

The part of a style run that exists on a single line is called a **text segment.** A larger division than a style run is the **font run,** which comprises those characters which share the same font. The font, style, size, and color aspects of text are collectively referred to as **character attributes.**

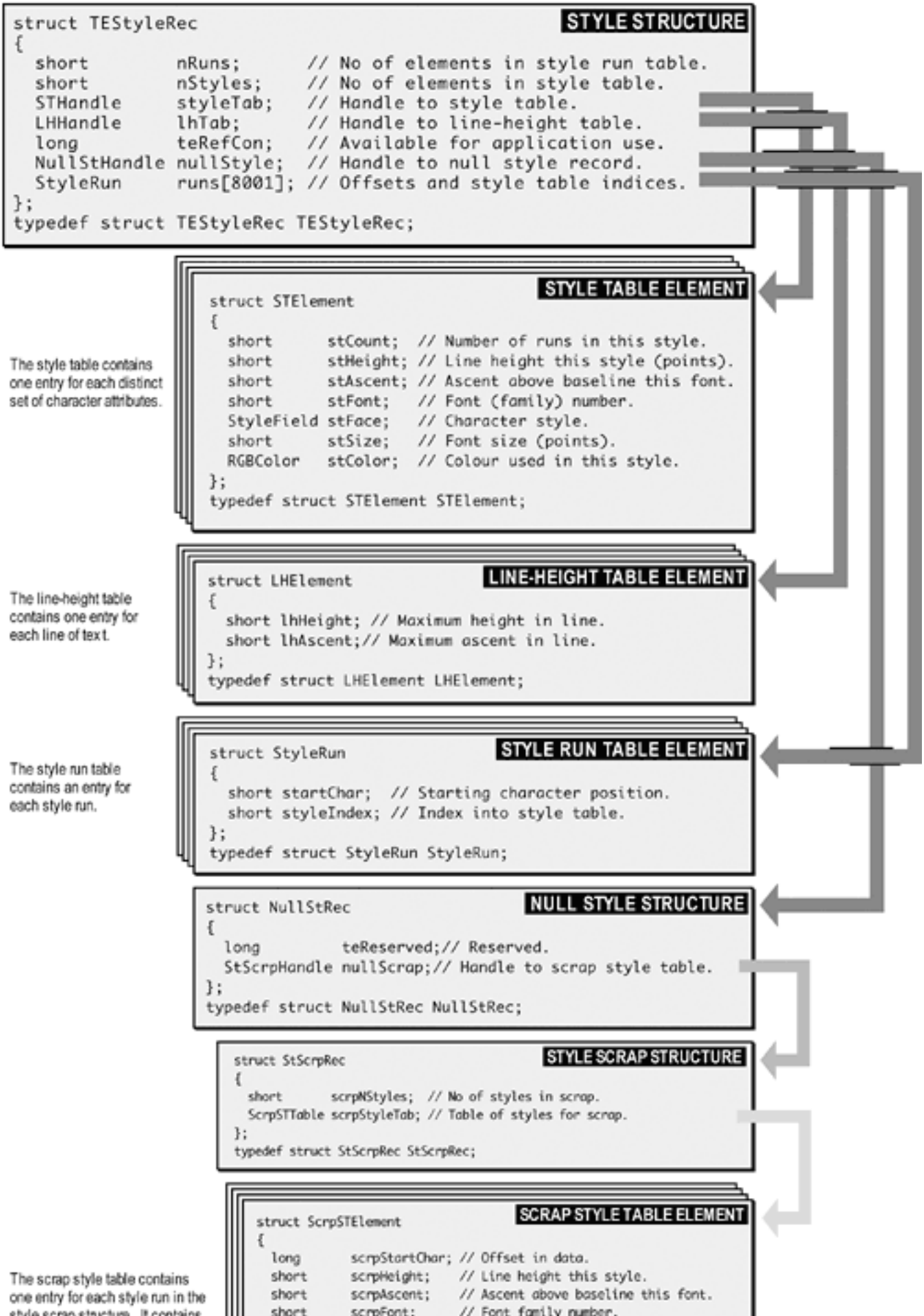
Additional TextEdit Data Structures for Multistyled Text

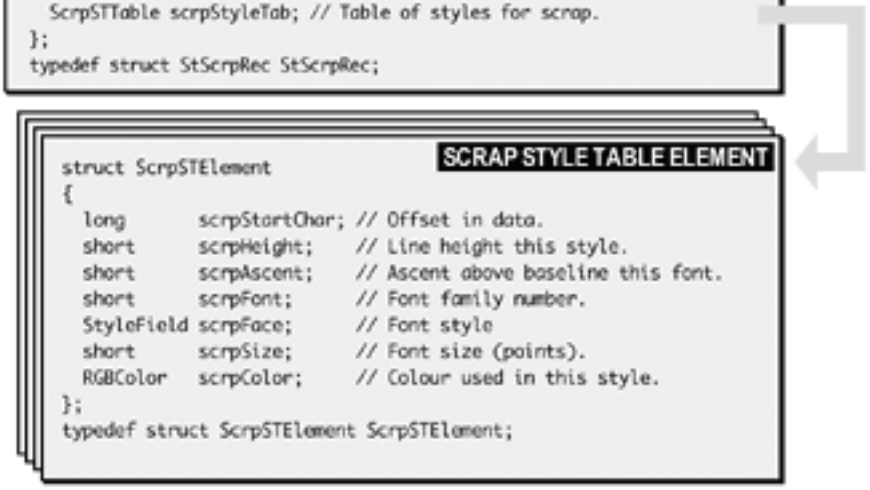
The TextEdit structure and the dispatch structure are the only data structures associated with monostyled text. However, when you allocate a multistyled TextEdit structure, a number of additional subsidiary data structures are created to support the text styling capabilities. The first of these additional data structures is the **style structure,** which stores the character attribute information for the text. (Recall that, when a multistyled TextEdit structure is created, the bytes at the `txFont`, `txFace`, and `filler` fields of the TextEdit structure contain a handle to the style structure.)

The additional data structures associated with a multistyled TextEdit structure are shown in Fig

6.

Figure 6. THE STYLE STRUCTURE AND SUBSIDIARY DATA STRUCTURES


STYLE STRUCTURE
struct TEStyleRec
{
short nRuns; // No of elements in style run table.
short nStyles; // No of elements in style table.
STHandle styleTab; // Handle to style table.
LHHandle lhTab; // Handle to line-height table.
long teRefCon; // Available for application use.
NullStHandle nullStyle; // Handle to null style record.
StyleRun runs[8001]; // Offsets and style table indices.
};
typedef struct TEStyleRec TEStyleRec;
STYLE TABLE ELEMENT
struct STElement
{
short stCount; // Number of runs in this style.
short stHeight; // Line height this style (points).
short stAscent; // Ascent above baseline this font.
short stFont; // Font (family) number.
StyleField stFace; // Character style.
short stSize; // Font size (points).
RGBColor stColor; // Colour used in this style.
};
typedef struct STElement STElement;
The style table contains one entry for each distinct set of character attributes.
LINE-HEIGHT TABLE ELEMENT
struct LHElement
{
short lhHeight; // Maximum height in line.
short lhAscent;// Maximum ascent in line.
};
typedef struct LHElement LHElement;
The line-height table contains one entry for each line of text.
STYLE RUN TABLE ELEMENT
struct StyleRun
{
short startChar; // Starting character position.
short styleIndex; // Index into style table.
};
typedef struct StyleRun StyleRun;
The style run table contains an entry for each style run.
NULL STYLE STRUCTURE
struct NullStRec
{
long teReserved;// Reserved.
StScrpHandle nullScrap;// Handle to scrap style table.
};
typedef struct NullStRec NullStRec;
STYLE SCRAP STRUCTURE
struct StScrpRec
{
short scrpNStyles; // No of styles in scrap.
ScrpSTTable scrpStyleTab; // Table of styles for scrap.
};
typedef struct StScrpRec StScrpRec;
SCRAP STYLE TABLE ELEMENT
struct ScrpSTElement
{
long scrpStartChar; // Offset in data.
short scrpHeight; // Line height this style.
short scrpAscent; // Ascent above baseline this font.
short scrpFont; // Font family number.
The scrap style table contains one entry for each style run in the


ScrpSTTable scrpStyleTab; // Table of styles for scrap.
};
typedef struct StScrpRec StScrpRec;
SCRAP STYLE TABLE ELEMENT
struct ScrpSTElement
{
long scrpStartChar; // Offset in data.
short scrpHeight; // Line height this style.
short scrpAscent; // Ascent above baseline this font.
short scrpFont; // Font family number.
StyleField scrpFace; // Font style
short scrpSize; // Font size (points).
RGBColor scrpColor; // Colour used in this style.
};
typedef struct ScrpSTElement ScrpSTElement;

The scrap style table contains one entry for each style run in the style scrap structure. It contains information similar to that contained in the style table.

Creating a Multistyled TextEdit Structure

The multistyled TextEdit structure is created by calling `TEStyleNew`.

Inserting Text

The following describes `TEStyleInsert`, which is used to insert multistyled text:

Function	*Use To*	*Comments*
`TEStyleInsert`	Insert multistyled text into the TextEdit structure immediately before the selection range or insertion point.	Does not affect the selection range. Redraws the text if necessary. Applies the specified character attributes to the text. (You should create your own style scrap structure, specifying the style attributes to be inserted and applied to the text. These attributes are copied directly into the style structure's style table.)

Formatting and Displaying Dates, Times, and Numbers

Preamble-The Text Utilities and International Resources

The Text Utilities

The **Text Utilities** are a collection of text-handling functions which you can use to, amongst other things, format numbers, currency, dates, and times.

International Resources

Many Text Utilities functions utilize the **international resources,** which define how different text elements are represented depending on the script system in use. The international resources relevant to formatting numbers, currency, dates, and times are as follows:

- ***Numeric Format Resource.*** The numeric format (`'itl0'`) resource contains short date and time formats, and formats for currency and numbers. It provides separators for decimals, thousands, and lists. It also contains the region code for this particular resource. Three of the several variations in short date and time formats are as follows:

System Software	*Morning*	*Afternoon*	*Short Date*
United States	`1:02 AM`	`1:02 PM`	`2/1/90`
Sweden	`01:02`	`13:02`	`90-01-01`
Germany	`1:02 Uhr`	`13:02 Uhr`	`2.1.1990`

- ***Long Date Format Resource.*** The long date format (`'itl1'`) resource specifies the long and abbreviated date formats for a particular region, including the names of days and months and the exact order of presentation of the elements. It also contains a region code for this particular resource. Three of the several variations of the long and abbreviated date formats are as follows:

System Software	*Abbreviated Date*	*Long Date*
United States	`Tue, Jan 2, 1990`	`Tuesday, January 2 1990`
French	`Mar 2 Jan 1990`	`Mardi 2 Janvier 1990`
Australian	`Tue, 2 Jan 1990`	`Tuesday, 2 January 1990`

- ***Tokens Resource.*** The tokens (`'itl4'`) resource contains, amongst other things, a table for formatting numbers. This table, which is called the **number parts table,** contains standard representations for the components of numbers and numeric strings. As will be seen, certain Text Utilities number formatting functions use the number parts table to create number strings in localized formats.

Date and Time

The Text Utilities functions that work with dates and times use information in the international resources to create different representations of date and time values. The Operating System provides functions that return the current date and time in numeric format. Text Utilities functions can then be used to convert these values into strings which can, in turn, be presented in the different international formats.

Date and Time Value Representations

The Operating System provides the following differing representations of date and time values:

Representation	*Description*
Standard date-time value.	A 32-bit integer representing the number of seconds between midnight, 1 January 1904 and the current time.
Long date-time value.	A 64-bit signed representation of data type `LongDateTime`. Allows for coverage of a longer time span than the standard date-time value, specifically, about 30,000 years.
Date-time structure.	Data type `DateTimeRec`. Includes integer fields for year, month, day, hour, minute, second, and day of week.
Long date-time structure.	Data type `LongDateRec`. Similar to the date-time structure, except that it adds several additional fields, including integer values for the era, day of the year, and week of the year. Allows for a longer time span than the date-time structure.

The date-time (`DateTimeRec`) and the long date-time (`LongDateRec`) structures are as follows:

```
struct DateTimeRec
{
  short year;
  short month;
  short day;
  short hour;
  short minute;
  short second;
  short dayOfWeek;
} ;

typedef struct DateTimeRec DateTimeRec;
```

```
union LongDateRec
{
  struct
  {
     short era;
    short year;
    short month;
    short day;
    short hour;
    short minute;
    short second;
    short dayOfWeek;
     short dayOfYear;
    short weekOfYear;
    short pm;
    short res1;
    short res2;
    short res3;
  }  ld;
  short list[14];
  struct
  {
    short       eraAlt;
    DateTimeRec oldDate;
  }  od;
} ;

typedef union LongDateRec LongDateRec;
```

Obtaining Date-Time Values and Structures

The Operating System Utilities provide the following two functions for obtaining date-time values and structures.

Function	*Description*
`GetDateTime`	Returns a standard date-time value.
`GetTime`	Returns a date-time structure.

Converting Between Values and Structures

The Operating System provides the following four functions for converting between the different date and time data types:

Function	*Converts*	*To*
`DateToSeconds`	Date-time structure.	Standard date-time value.
`SecondsToDate`	Standard date-time value.	Date-time structure.
`LongDateToSeconds`	Long date-time structure.	Long date-time value.
`LongSecondsToDate`	Long date-time value.	Long date-time structure.

Converting Date-Time Values into Strings

The Text Utilities provide the following functions for converting from one of the numeric date-time representations to a formatted string.

Function	*Description*
`DateString`	Converts standard date-time value to a date string formatted according to the specified international resource.
`LongDateString`	Converts long date-time value to a date string formatted according to the specified international resource.
`TimeString`	Converts standard date-time value to a time string formatted according to the specified international resource.
`LongTimeString`	Converts long date-time values to a time string formatted according to the specified international resource.

Output Format-Date. When you use `DateString` and `LongDateString`, you can specify, in the `longFlag` parameter, an output format for the resulting date string. This format can be one of the following three values of the `DateForm` enumerated data type:

Value	*Date String Produced (Example)*	*Formatting Information Obtained From*
`shortDate`	`1/31/92`	`Numeric format resource ('itl0')`.
`abbrevDate`	`Fri, Jan 31, 1992`	`Long date format resource ('itl1')`.
`longDate`	`Friday, January 31, 1992`	`Long date format resource ('itl1')`.

Output Format-Time. When you use `TimeString` and `LongTimeString`, you can request an output format for the resulting time string by specifying either `true` or `false` in the `wantSeconds` parameter. `true` will cause seconds to be included in the string.

`DateString`, `LongDateString`, `TimeString` and `LongTimeString` use the date and time formatting information in the format resource that you specify in the resource handle (`intlHandle`) parameter. If you specify `NULL` for the value of the resource handle parameter, the appropriate format resource for the current script system is used.

Converting Date-Time Strings into Internal Numeric Representation

The Text Utilities include functions that can parse date and time strings as entered by users and fill in the fields of a structure with the components of the date and time, including the month, day, year, hours, minutes, and seconds, extracted from the string.

Suppose your application needs to convert a date and time string typed in by the user (for example, "`March 27, 1992, 08:14 p.m`.") into numeric representation. The following Text Utilities functions can be used to convert the string entered by the user into a long date-time structure:

Function	*Description*
`StringToDate`	Parses an input string for a date and creates an internal numeric representation of that date. Returns a status value indicating the confidence level for the success of the conversion. Expects a date specification, in one of the formats defined by the current script system, at the beginning of the string. Recognizes date strings in many formats, for example: "`September 1,1987`", "`1 Sept 87`", "`1/9/87`", and "`1 1987 Sept`".
`StringToTime`	Parses an input string for a time and creates an internal numeric representation of that time. Returns a status value indicating the confidence level for the success of the conversion. Expects a time specification, in a format defined by the current script system, at the beginning of the string.

You usually call `StringToDate` and `StringToTime` sequentially to parse the date and time values from an input string and fill in the fields of a long date-time structure. Note that `StringToDate` assigns to its `lengthUsed` parameter the number of bytes that it uses to parse the date. Use this value to compute the starting location of the text that you can pass to `StringToTime`.

The "confidence level" value returned by both `StringToDate` and `StringToTime` is of type `StringToDateStatus`, a set of bit values which have been OR'd together. The higher the resultant number, the lower the confidence level. Three of the twelve `StringToDateStatus` values, and their meanings, are as follows:

Value	*Meaning*
`fataldateTime`	Fatal error during the parse.
`dateTimeNotFound`	Valid date or time value not be found in string.
`sepNotIntlSep`	Valid date or time value found, but one or more of the separator characters in the string was not an expected separator character for the script system in use.

Date Cache Structure. Both `StringToDate` and `StringToTime` take a **date cache structure** as one of their parameters. A date cache structure (a data structure of type `DateCacheRec`) stores date conversion data used by the date and time conversion functions. You must declare a data cache structure in your application and initialize it by calling `InitDateCache` once, typically in your main program initialization code.

Numbers

The Text Utilities provide several functions for converting between the internal numeric representation of a number and the output (or input) format of that number. You will need to perform these conversions when the user enters numbers for your application to use or when you present numbers to the user.

Integers

The simplest number conversion tasks involve integer values. The following Text Utilities functions may be used to convert an integer value to a numeric string and vice versa:

Function	*Description*
`NumToString`	Converts a long integer value into a string representation.
`StringToNum`	Converts a string representation of a number into a long integer value.

The range of values accommodated by these functions is -2,147,483,647 to 2,147,483,648. No comma insertion or other formatting is performed.

Number Format Specification Strings

Number format specification strings define the appearance of numeric strings. When you need to accommodate the differences in number output formats for different countries and regions, or when you are working with floating point numbers, you will need to use number format specification strings.

Parts. Each number format specification string contains up to three parts:

- The positive number format.
- The negative number format.
- The zero number format.

Each of these formats is applied to a numeric value of the corresponding type. When the specification string contains only one part, that part is used for all values. When in contains two parts, the first part is used for positive and zero values and the second part is used for negative values.

Elements. A number format specification string can contain the following elements:

- Number parts separators (, and .) for the decimal separator and the thousands separator.
- Literals to be included in the output. (Literals can be strings or brackets, braces and parentheses, and must be enclosed in quotation marks.)
- Digit place holders. (Digit place holders that you want displayed must be indicated by digit symbols. Zero digits (`0`) add leading zeroes whenever an input digit is not present. Skipping digits (`#`) only produce output characters when an input digit is present. Padding digits (`^`) are like zero digits except that a padding character such as a non-breaking space is used instead of leading zeros to pad the output string.)
- Quoting mechanisms for handling literals correctly.
- Symbol and sign characters.

Examples. The following shows several different number format specification strings and the output produced by each:

Number Format Specification String	*Numeric Value*	*Output Format*
`###,###.##;-###,###.##;0`	876543.21	`876,543.21`
`###,###.0##,###`	4321	`4,321.0`
`###,###.0##,###`	7.563489	`7.563,489`
`###;(000);^^^`	-1	`(001)`
`###.###`	5.234999	`5.235`
`###'CR';###'DB';''zero''`	1	`1CR`

`###'CR';###'DB';''zero''`	0	`'zero'`
`##%`	0.1	`10%`

Integer digits are always filled in from the right and decimal places are always filled in from the left. The following examples, in which a literal is included in the middle of the format strings, demonstrate this behavior:

Number Format Specification String	*Numeric Value*	*Output Format*
`###'ab'###`	1	`1`
`###'ab'###`	123	`123`
`###'ab'###`	1234	`1ab1234`
`0.###'ab'###`	0.1	`0.1`
`0.###'ab'###`	0.123	`1.123`
`0.###'ab'###`	0.1234	`0.123ab4`

Overflow and Rounding. If the input string contains more digits than are specified in the number format specification string, an error (`formatOverflow`) will be generated. If the input string contains more decimal places than are specified in the number format specification string, the decimal portion is automatically rounded.

Converting Number Format Specification Strings to Internal Numeric Representations. With the required number format specification string defined, you must then convert the string into an internal numeric representation. The internal representation of format strings is stored in a `NumFormatString` structure. You use the following functions to convert a number format specification string to a `NumFormatString` structure and vice versa.

Function	*Description*
`StringToFormatRec`	Converts a number format specification string into a `NumFormatString` structure.
`FormatRecToString`	Convert a `NumFormatString` structure back to a number format specification string.

Number Parts Table. The internal numeric representation allows you to map the number into different output formats. One of the parameters taken by `StringToFormatRec` is a number parts table. The number parts table specifies which characters are used for certain purposes, such as separating parts of a number, in the format specification string. (For example, a thousands separator is a comma in Australia and a decimal point in France.) As previously stated, the number parts table is contained in the `'itl4'` resource. A handle to the `'itl4'` resource may be obtained by a call to `GetIntlResourceTable`, specifying `iuNumberPartsTable` in the `tableCode` parameter.

> **Note**
>
> The `FormatRecToString` function also contains a number parts table parameter. By using a different table than was used in the call to `StringToFormatRec`, you can produce a number format specification string that specifies how numbers are formatted for a different region of the world. You use `FormatRecToString` when you want to display the number format specification string to the user for perusal or modification.

Converting Between Floating Point Numbers and Numeric Strings

Armed with a `NumFormatString` structure, you can convert floating point numbers into numeric strings and numeric strings into floating point numbers using the following functions:

Function	*Description*
`StringToExtended`	Using a `NumFormatString` structure and a number parts table, converts a numeric string to an 80-bit floating point value.
`ExtendedToString`	Using a `NumFormatString` structure and a number parts table, converts an 80-bit floating point number to a numeric string.

PowerPC Considerations. The PowerPC-based Macintosh follows the IEEE 754 standard for floating point arithmetic. In this standard, `float` is 32 bits and `double` is 64 bits. (Apple has added the 128-bit `long double` type.) However, the PowerPC floating point unit does not support Motorola's 80/96-bit `extended` type, and neither do the PowerPC numerics. To accommodate this, you can use Apple-supplied conversion utilities to move to and from `extended`. For example, the functions `x80tod` and `dtox80` (see the header file fp.h) can be used to directly transform 680x0 80-bit `extended` data types to `double` and back.

`StringToFormatRec`, `FormatRecToString`, `StringToExtended`, and `ExtendedToString` return a result of type `FormatStatus`, which is an integer value. The low byte is of type `FormatResultType`. Typical examples of the returned format status are as follows:

Value	*Meaning*
`fFormatOK`	The format of the input value is appropriate and the conversion was successful.
`fBestGuess`	The format of the input value is questionable. The result of the conversion may or may not be correct.
`fBadPartsTable`	The parts table is not valid.

Main TextEdit Constants, Data Types and Functions

Constants

Alignment

```
teFlushDefault           = 0
teCenter                 = 1
teFlushRight             = -1
teFlushLeft              = -2
```

Feature or Bit Definitions for TEFeatureFlag Feature Parameter

```
teFAutoScroll            = 0
teFTextBuffering         = 1
teFOutlineHilite         = 2
teFInlineInput           = 3
teFUseWhiteBackground    = 4
teFUseInlineInput        = 5
teFInlineInputAutoScroll = 6
```

Data Types

```
typedef char     Chars[32001];
typedef char     *CharsPtr;
typedef CharsPtr *CharsHandle;
```

TextEdit Structure

```
struct TERec
{
  Rect           destRect;      // Destination rectangle.
  Rect           viewRect;      // View rectangle.
  Rect           selRect;       // Selection rectangle.
  short          lineHeight;    // Vert spacing of lines. -1 in multistyled edit struct.
  short          fontAscent;    // Font ascent. -1 in multistyled TextEdit structure.
  Point          selPoint;      // Point selected with the mouse.
  short          selStart;      // Start of selection range.
  short          selEnd;        // End of selection range.
  short          active;        // Set when structure is activated or deactivated.
  WordBreakUPP   wordBreak;     // Word break function.
  TEClickLoopUPP clickLoop;     // Click loop function.
  long           clickTime;     // (Used internally.)
  short          clickLoc;      // (Used internally.)
  long           caretTime;     // (Used internally.)
  short          caretState;    // (Used internally.)
  short          just;          // Text alignment.
  short          teLength;      // Length of text.
  Handle         hText;         // Handle to text to be edited.
  long           hDispatchRec;  // Handle to TextEdit dispatch structure.
  short          clikStuff;     // (Used internally)
  short          crOnly;        // If < 0, new line at Return only.
  short          txFont;        // Text font.  // If multistyled edit struct (txSize = -1),
  StyleField     txFace;        // Chara style. // these bytes are used as a handle
  SInt8          filler;        //              // to a style structure (TEStyleHandle).
  short          txMode;        // Pen mode.
  short          txSize;        // Font size. -1 in multistyled TextEdit structure.
  GrafPtr        inPort;        // Pointer to grafPort for this TextEdit structure.
  HighHookUPP    highHook;      // Used for text highlighting, caret appearance.
```

```
  CaretHookUPP   caretHook;     // Used from assembly language.
  short          nLines;        // Number of lines.
  short          lineStarts[16001];  // Positions of line starts.
} ;
typedef struct TERec TERec;
typedef TERec *TEPtr;
typedef TEPtr *TEHandle;
```

Style Structure

```
struct TEStyleRec
{
  short        nRuns;          // Number of style runs.
  short        nStyles;        // Size of style table.
  STHandle     styleTab;       // Handle to style table.
  LHHandle     lhTab;          // Handle to line-height table.
  long         teRefCon;       // Reserved for application use.
  NullStHandle nullStyle;      // Handle to style set at null selection.
  StyleRun  runs[8001];        // ARRAY [0..8000] OF StyleRun.
} ;
typedef struct TEStyleRec TEStyleRec;
typedef TEStyleRec *TEStylePtr;
typedef TEStylePtr *TEStyleHandle;
```

Text Style Structure

```
struct TextStyle
{
  short        tsFont;         // Font (family) number.
  StyleField   tsFace;         // Character Style.
  short        tsSize;         // Size in point.
  RGBColor     tsColor;        // Absolute (RGB) color.
} ;
typedef struct TextStyle TextStyle;
typedef TextStyle *TextStylePtr;
typedef TextStylePtr *TextStyleHandle;
```

Functions

Creating and Disposing of TextEdit Structures

```
TEHandle    TENew(const Rect *destRect,const Rect *viewRect);
TEHandle    TEStyleNew(const Rect *destRect,const Rect *viewRect);
void        TEDispose(TEHandle hTE);
```

Activating and Deactivating a TextEdit Structure

```
void        TEActivate(TEHandle hTE);
void        TEDeactivate(TEHandle hTE);
```

Setting and Getting aTextEdit Structure's Text

```
void        TEKey(short key,TEHandle hTE);
void        TESetText(const void *text,long length,TEHandle hTE);
CharsHandle TEGetText(TEHandle hTE);
```

Setting the Caret and Selection Range

```
void        TEIdle(TEHandle hTE);
void        TEClick(Point pt,Boolean fExtend,TEHandle h);
void        TESetSelect(long selStart,long selEnd,TEHandle hTE);
```

Displaying and Scrolling Text

```
void        TESetAlignment(short just,TEHandle hTE);
void        TEUpdate(const Rect *rUpdate,TEHandle hTE);
void        TETextBox(const void *text,long length,const Rect *box,short just);
void        TECalText(TEHandle hTE);
long        TEGetHeight(long endLine,long startLine,TEHandle hTE);
void        TEScroll(short dh,short dv,TEHandle hTE);
void        TEPinScroll(short dh,short dv,TEHandle hTE);
void        TEAutoView(Boolean fAuto,TEHandle hTE);
void        TESelView(TEHandle hTE);
```

Modifying the Text of a TextEdit Structure

```
void        TEDelete(TEHandle hTE);
void        TEInsert(const void *text,long length,TEHandle hTE);
void        TEStyleInsert(const void *text,long length,StScrpHandle hST,TEHandle hTE);
void        TECut(TEHandle hTE);
void        TECopy(TEHandle hTE);
void        TEPaste(TEHandle hTE);
OSErr       TEFromScrap(void);
OSErr       TEToScrap(void);
```

Managing the TextEdit Private Scrap

```
Handle      TEScrapHandle(void);
long        TEGetScrapLength(void);
void        TESetScrapLength(long length);
```

Using Byte Offsets and Corresponding Points

```
short       TEGetOffset(Point pt,TEHandle hTE);
Point       TEGetPoint(short offset,TEHandle hTE);
```

Customising TextEdit

```
void        TESetClickLoop(TEClickLoopUPP clikProc,TEHandle hTE);;
void        TESetWordBreak(WordBreakUPP wBrkProc,TEHandle hTE);;
void        TECustomHook(TEIntHook which, UniversalProcPtr *addr,TEHandle hTE);
```

Additional TextEdit Features

```
short       TEFeatureFlag(short feature,short action,TEHandle hTE);
```

Main Constants, Data Types and Functions Relating to Dates, Times and Numbers

Constants

StringToDate and StringToTime Status Values

```
fatalDateTime      = 0x8000  Mask to a fatal error.
longDateFound      = 1       Mask to long date found.
leftOverChars      = 2       Mask to warn of left over characters.
sepNotIntlSep      = 4       Mask to warn of non-standard separators.
fieldOrderNotIntl  = 8       Mask to warn of non-standard field order.
extraneousStrings  = 16      Mask to warn of unparsable strings in text.
tooManySeps        = 32      Mask to warn of too many separators.
sepNotConsistent   = 64      Mask to warn of inconsistent separators.
tokenErr           = 0x8100  Mask for 'tokenizer err encountered'.
cantReadUtilities  = 0x8200
dateTimeNotFound   = 0x8400
dateTimeInvalid    = 0x8800
```

FormatResultType Values for Numeric Conversion Functions

```
fFormatOK          = 0
fBestGuess         = 1
fOutOfSynch        = 2
fSpuriousChars     = 3
fMissingDelimiter  = 4
fExtraDecimal      = 5
fMissingLiteral    = 6
fExtraExp          = 7
fFormatOverflow    = 8
fFormStrIsNAN      = 9
fBadPartsTable     = 10
fExtraPercent      = 11
fExtraSeparator    = 12
fEmptyFormatString = 13
```

Data Types

```
typedef short  StringToDateStatus;
typedef SInt8  DateForm;
typedef short  FormatStatus;
typedef Sint8  FormatResultType;
```

Data Cache Structure

```
struct DateCacheRecord
{
  short  hidden[256];  // Only for temporary use.
} ;
typedef struct DateCacheRecord DateCacheRecord;
typedef DateCacheRecord *DateCachePtr;
```

Number Format Specification Structure

```
struct NumFormatString
{
  UInt8 fLength;
  UInt8 fVersion;
```

```
  char  data[254];    // Private data.
} ;
typedef struct NumFormatString NumFormatString;
typedef NumFormatString NumFormatStringRec;
```

Functions

Getting Date-Time Values and Structures

```
void    GetDateTime(unsigned long *secs);
void    GetTime(DateTimeRec *d);
```

Converting Between Date-Time Values and Structures

```
void    DateToSeconds(const DateTimeRec *d,unsigned long *secs);
void    SecondsToDate(unsigned long secs,DateTimeRec *d);
void    LongDateToSeconds(const LongDateRec *lDate,LongDateTime *lSecs);
void    LongSecondsToDate(LongDateTime *lSecs,LongDateRec *lDate);
```

Converting Date-Time Strings into Internal Numeric Representation

```
OSErr   InitDateCache(DateCachePtr theCache);
StringToDateStatus  StringToDate(Ptr textPtr,long textLen,DateCachePtr theCache,
                    long *lengthUsed,LongDateRec *dateTime);
StringToDateStatus  StringToTime(Ptr textPtr,long textLen,DateCachePtr theCache,
                    long *lengthUsed,LongDateRec *dateTime);
```

Converting Long Date and Time Values into Strings

```
void    DateString(long dateTime,DateForm longFlag,Str255 result);
void    TimeString(long dateTime,Boolean wantSeconds,Str255 result);
void    LongDateString(LongDateTime *dateTime,DateForm longFlag,Str255 result,
        Handle intlHandle);
void    LongTimeString(LongDateTime *dateTime,Boolean wantSeconds,Str255 result,
        Handle intlHandle);
```

Converting Between Integers and Strings

```
void    StringToNum(ConstStr255Param theString,long *theNum);
void    NumToString(long theNum,Str255 theString);
```

Using Number Format Specification Strings for International Number Formatting

```
FormatStatus  StringToFormatRec(ConstStr255Param inString,const NumberParts *partsTable,
              NumFormatString *outString)
FormatStatus  FormatRecToString(const NumFormatString *myCanonical,
              const NumberParts *partsTable,Str255 outString,TripleInt positions)
```

Converting Between Strings and Floating Point Numbers

```
FormatStatus  ExtendedToString(const extended80 x,const NumFormatString *myCanonical,
              const NumberParts *partsTable,Str255 outString)
FormatStatus  StringToExtended(ConstStr255Param source,const NumFormatString *myCanonical,
              const NumberParts *partsTable,extended80 *x)
```

Moving to and from Extended

```
void    x80told(const extended80 *x80,long double *x);
void    ldtox80(const long double *x,extended80 *x80);
double  x80tod(const extended80 *x80);
```

```
void    dtox80(const double *x, extended80 *x80);
```

Demonstration Program MonoTextEdit Listing

```
// ****************************************************************************************
// MonoTextEdit.c                                                          CARBON EVENT MODEL
// ****************************************************************************************
//
// This program demonstrates:
//
// •  A "bare-bones" monostyled text editor.
//
// •  A Help dialog which features the integrated scrolling of multistyled text and pictures.
//
// In the monostyled text editor demonstration, a panel is displayed at the bottom of all
// opened windows. This panel displays the edit record length, number of lines, line height,
// destination rectangle (top), scroll bar/scroller value, and scroll bar/scroller maximum
// value.
//
// The bulk of the source code for the Help dialog is contained in the file HelpDialog.c.
// The dialog itself displays information intended to assist the user in adapting the Help
// dialog source code and resources to the requirements of his/her own application.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  An 'MBAR' resource, and 'MENU' resources for Apple, File, Edit, and Help dialog pop-up
//    menus (preload, non-purgeable).
//
// •  A 'CNTL' resources (purgeable) for the vertical scroll bar in the text editor window.
//
// •  'TEXT' and associated 'styl' resources (all purgeable) for the Help dialog.
//
// •  'PICT' resources (purgeable) for the Help dialog.
//
// •  A 'STR#' resource  (purgeable) containing error text strings.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//

// ****************************************************************************************

// ..................................................................................... includes

#include <Carbon.h>

// .....................................................................................  defines

#define rMenubar          128
#define mAppleApplication 128
#define  iAbout           1
#define  iHelp            2
#define mFile             129
#define  iNew             1
#define  iOpen            2
#define  iClose           4
#define  iSaveAs          6
#define  iQuit            12
#define mEdit             130
#define  iUndo            1
#define  iCut             3
#define  iCopy            4
#define  iPaste           5
```

```
#define  iClear           6
#define  iSelectAll       7
#define rVScrollbar       128
#define rErrorStrings     128
#define  eMenuBar         1
#define  eWindow          2
#define  eDocStructure    3
#define  eEditRecord      4
#define  eExceedChara     5
#define  eNoSpaceCut      6
#define  eNoSpacePaste    7
#define kMaxTELength      32767
#define kTab              0x09
#define kBackSpace        0x08
#define kForwardDelete    0x7F
#define kReturn           0x0D
#define kEscape           0x1B
#define topLeft(r)        (((Point *) &(r))[0])
#define botRight(r)       (((Point *) &(r))[1])

// ..............................................................................
typedefs

typedef struct
{
  TEHandle   textEditStrucHdl;
  ControlRef vScrollbarRef;
}  docStructure, **docStructureHandle;
// ..................................................................... global
variables

Boolean          gRunningOnX = false;
MenuID           gHelpMenu;
ControlActionUPP gScrollActionFunctionUPP;
TEClickLoopUPP   gCustomClickLoopUPP;
SInt16           gNumberOfWindows = 0;
SInt16           gOldControlValue;

// ..................................................................... function
prototypes

void            main                 (void);
void            doPreliminaries      (void);
OSStatus        appEventHandler      (EventHandlerCallRef,EventRef,void *);
OSStatus        windowEventHandler   (EventHandlerCallRef,EventRef,void *);
void            doIdle               (void);
void            doKeyEvent           (SInt8);
void            scrollActionFunction (ControlRef,SInt16);
void            doInContent          (Point,Boolean);
void            doDrawContent        (WindowPtr);
void            doActivateDeactivate (WindowRef,Boolean);
WindowRef       doNewDocWindow       (void);
EventHandlerUPP doGetHandlerUPP      (void);
Boolean         customClickLoop      (void);
void            doSetScrollBarValue  (ControlRef,SInt16 *);
void            doAdjustMenus        (void);
void            doMenuChoice         (MenuID,MenuItemIndex);
void            doFileMenu           (MenuItemIndex);
void            doEditMenu           (MenuItemIndex);
SInt16          doGetSelectLength    (TEHandle);
void            doAdjustScrollbar    (WindowRef);
void            doAdjustCursor       (WindowRef);
void            doCloseWindow        (WindowRef);
void            doSaveAsFile         (TEHandle);
void            doOpenCommand        (void);
void            doOpenFile           (FSSpec);
void            doDrawDataPanel      (WindowRef);
```

```
void            doErrorAlert        (SInt16);
void            navEventFunction    (NavEventCallbackMessage,NavCBRecPtr,
                                     NavCallBackUserData);

extern void     doHelp              (void);

// ********************************************************************************** main

void  main(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  EventTypeSpec applicationEvents[] = {  {  kEventClassApplication, kEventAppActivated    } ,
                                         {  kEventClassCommand,     kEventProcessCommand  } ,
                                         {  kEventClassMenu,        kEventMenuEnableItems } ,
                                         {  kEventClassMouse,
kEventMouseMoved      }  } ;
  // ................. ..................................................... do
preliminaries

  doPreliminaries();

  // ..................................................................... set up menu bar and
menus

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    doErrorAlert(eMenuBar);
  SetMenuBar(menubarHdl);
  DrawMenuBar();

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
    }

    menuRef = GetMenuRef(mAppleApplication);
    DeleteMenuItem(menuRef,iHelp);

    HMGetHelpMenu(&menuRef,NULL);
    InsertMenuItem(menuRef,"\pMonoTextEdit Help",0);
    gHelpMenu = GetMenuID(menuRef);

    gRunningOnX = true;
  }
  else
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
      SetMenuItemCommandID(menuRef,iQuit,kHICommandQuit);
  }

  // ..................................................... create universal procedure
pointers

  gScrollActionFunctionUPP = NewControlActionUPP((ControlActionProcPtr) scrollActionFunction);
  gCustomClickLoopUPP      = NewTEClickLoopUPP((TEClickLoopProcPtr) customClickLoop);

  // ..................................................... install application event
handler
```

```
  InstallApplicationEventHandler(NewEventHandlerUPP((EventHandlerProcPtr) appEventHandler),
                                 GetEventTypeCount(applicationEvents),applicationEvents,
                                 0,NULL);

  // ..................................................................... install a
timer

  InstallEventLoopTimer(GetCurrentEventLoop(),0,TicksToEventTime(GetCaretTime()),
                        NewEventLoopTimerUPP((EventLoopTimerProcPtr) doIdle),NULL,
                        NULL);
  // ..................................................................... open an untitled
window

  doNewDocWindow();

  // ..................................................................... run application
event loop

  RunApplicationEventLoop();
}

// ********************************************************************************** doPreliminaries

void  doPreliminaries(void)
{
  MoreMasterPointers(192);
  InitCursor();
}

// ************************************************************************************** appEventHandler

OSStatus  appEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                          void * userData)
{
  OSStatus      result = eventNotHandledErr;
  UInt32        eventClass;
  UInt32        eventKind;
  HICommand     hiCommand;
  MenuID        menuID;
  MenuItemIndex menuItem;

  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);

  switch(eventClass)
  {
    case kEventClassApplication:
      if(eventKind == kEventAppActivated)
        SetThemeCursor(kThemeArrowCursor);
      break;

    case kEventClassCommand:
      if(eventKind == kEventProcessCommand)
      {
        GetEventParameter(eventRef,kEventParamDirectObject,typeHICommand,NULL,
                          sizeof(HICommand),NULL,&hiCommand);
        menuID = GetMenuID(hiCommand.menu.menuRef);
        menuItem = hiCommand.menu.menuItemIndex;
        if((hiCommand.commandID != kHICommandQuit) &&
           ((menuID >= mAppleApplication && menuID <= mEdit)) || menuID == gHelpMenu)
        {
          doMenuChoice(menuID,menuItem);
          result = noErr;
        }
      }
      break;
    case kEventClassMenu:
```

```
      if(eventKind == kEventMenuEnableItems)
      {
        GetWindowClass(FrontWindow(),&windowClass);
        if(windowClass == kDocumentWindowClass)
          doAdjustMenus();
        result = noErr;
      }
      break;

    case kEventClassMouse:
      if(eventKind == kEventMouseMoved)
      {
        GetWindowClass(FrontWindow(),&windowClass);
        if(windowClass == kDocumentWindowClass)
          doAdjustCursor(FrontWindow());
       result = noErr;
      }
      break;

  return result;
}

// ********************************************************************** windowEventHandler

OSStatus  windowEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                             void* userData)
{
  OSStatus        result = eventNotHandledErr;
  UInt32          eventClass;
  UInt32          eventKind;
  WindowRef       windowRef;
  UInt32          modifiers;
  Point           mouseLocation;
  Boolean         shiftKeyDown = false;
  ControlRef      controlRef;
  ControlPartCode controlPartCode;
  SInt8           charCode;

  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);

  switch(eventClass)
  {
    case kEventClassWindow:                                                 // event class window
      GetEventParameter(eventRef,kEventParamDirectObject,typeWindowRef,NULL,sizeof(windowRef),
                        NULL,&windowRef);
      switch(eventKind)
      {
        case kEventWindowDrawContent:
          doDrawContent(windowRef);
          result = noErr;
          break;
        case kEventWindowActivated:
          doActivateDeactivate(windowRef,true);
          result = noErr;
          break;

        case kEventWindowDeactivated:
          doActivateDeactivate(windowRef,false);
          result = noErr;
          break;

        case kEventWindowClickContentRgn:
          GetEventParameter(eventRef,kEventParamMouseLocation,typeQDPoint,NULL,
                            sizeof(mouseLocation),NULL,&mouseLocation);
          SetPortWindowPort(FrontWindow());
          GlobalToLocal(&mouseLocation);
```

```
          GetEventParameter(eventRef,kEventParamKeyModifiers,typeUInt32,NULL,
                            sizeof(modifiers),NULL,&modifiers);
          if(modifiers & shiftKey)
            shiftKeyDown = true;
          doInContent(mouseLocation,shiftKeyDown);
          result = noErr;
          break;

        case kEventWindowClose:
          doCloseWindow(windowRef);
          result = noErr;
          break;
      }
      break;

    case kEventClassMouse:                                          // event class mouse
      switch(eventKind)
      {
        case kEventMouseDown:
          GetEventParameter(eventRef,kEventParamMouseLocation,typeQDPoint,NULL,
                            sizeof(mouseLocation),NULL,&mouseLocation);
          SetPortWindowPort(FrontWindow());
          GlobalToLocal(&mouseLocation);
          controlRef = FindControlUnderMouse(mouseLocation,FrontWindow(),&controlPartCode);
          if(controlRef)
          {
            gOldControlValue = GetControlValue(controlRef);
            TrackControl(controlRef,mouseLocation,gScrollActionFunctionUPP);
            result = noErr;
          }
        break;
      }
      break;

    case kEventClassKeyboard:                                       // event class keyboard
      switch(eventKind)
      {
        case kEventRawKeyDown:
        case kEventRawKeyRepeat:
          GetEventParameter(eventRef,kEventParamKeyMacCharCodes,typeChar,NULL,
                            sizeof(charCode),NULL,&charCode);
          GetEventParameter(eventRef,kEventParamKeyModifiers,typeUInt32,NULL,
                            sizeof(modifiers),NULL,&modifiers);
          if((modifiers & cmdKey) == 0)
            doKeyEvent(charCode);
          result = noErr;
          break;
      }
      break;
  }

  return result;
}

// ********************************************************************************** doIdle

void  doIdle(void)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;

  windowRef = FrontWindow();
  if(GetWindowKind(windowRef) == kApplicationWindowKind)
  {
    docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));
    if(docStrucHdl != NULL)
      TEIdle((*docStrucHdl)->textEditStrucHdl);
```

```
  }
}

// ********************************************************************************** doKeyEvent

void  doKeyEvent(SInt8 charCode)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt16             selectionLength;

  if(charCode <= kEscape && charCode != kBackSpace && charCode != kReturn)
    return;

  windowRef = FrontWindow();
  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  if(charCode == kTab)
  {
    // Do tab key handling here if required.
  }
  else if(charCode == kForwardDelete)
  {
    selectionLength = doGetSelectLength(textEditStrucHdl);
    if(selectionLength == 0)
      (*textEditStrucHdl)->selEnd += 1;
    TEDelete(textEditStrucHdl);
    doAdjustScrollbar(windowRef);
  }
  else
  {
    selectionLength = doGetSelectLength(textEditStrucHdl);
    if(((*textEditStrucHdl)->teLength - selectionLength + 1) < kMaxTELength)
    {
      TEKey(charCode,textEditStrucHdl);
      doAdjustScrollbar(windowRef);
    }
    else
      doErrorAlert(eExceedChara);
  }

  doDrawDataPanel(windowRef);
}

// ********************************************************************** scrollActionFunction

void  scrollActionFunction(ControlRef controlRef,SInt16 partCode)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt16             linesToScroll;
  SInt16             controlValue, controlMax;

  windowRef = GetControlOwner(controlRef);
  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));;
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  controlValue = GetControlValue(controlRef);
  controlMax = GetControlMaximum(controlRef);

  if(partCode)
  {
    if(partCode != kControlIndicatorPart)
    {
```

```
      switch(partCode)
      {
        case kControlUpButtonPart:
        case kControlDownButtonPart:
          linesToScroll = 1;
          break;

        case kControlPageUpPart:
        case kControlPageDownPart:
          linesToScroll = (((*textEditStrucHdl)->viewRect.bottom -
                            (*textEditStrucHdl)->viewRect.top) /
                            (*textEditStrucHdl)->lineHeight) - 1;
          break;
      }

      if((partCode == kControlDownButtonPart) || (partCode == kControlPageDownPart))
        linesToScroll = -linesToScroll;

      linesToScroll = controlValue - linesToScroll;
      if(linesToScroll < 0)
        linesToScroll = 0;
      else if(linesToScroll > controlMax)
        linesToScroll = controlMax;

      SetControlValue(controlRef,linesToScroll);

      linesToScroll = controlValue - linesToScroll;
    }
    else
    {
      linesToScroll = gOldControlValue - controlValue;
      gOldControlValue = controlValue;
    }

    if(linesToScroll != 0)
    {
      TEScroll(0,linesToScroll * (*textEditStrucHdl)->lineHeight,textEditStrucHdl);
      doDrawDataPanel(windowRef);
    }
  }
}

// ********************************************************************************** doInContent

void  doInContent(Point mouseLocation,Boolean shiftKeyDown)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;

  windowRef = FrontWindow();
  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  if(PtInRect(mouseLocation,&(*textEditStrucHdl)->viewRect))
    TEClick(mouseLocation,shiftKeyDown,textEditStrucHdl);
}
// ********************************************************************************** doDrawContent

void  doDrawContent(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  GrafPtr            oldPort;
  RgnHandle          visibleRegionHdl = NewRgn();
  Rect               portRect;
```

```
  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

  GetPortVisibleRegion(GetWindowPort(windowRef),visibleRegionHdl);
  EraseRgn(visibleRegionHdl);

  UpdateControls(windowRef,visibleRegionHdl);

  GetWindowPortBounds(windowRef,&portRect);
  TEUpdate(&portRect,textEditStrucHdl);

  doDrawDataPanel(windowRef);

  DisposeRgn(visibleRegionHdl);
  SetPort(oldPort);
}

// ********************************************************************** doActivateDocWindow

void  doActivateDeactivate(WindowRef windowRef,Boolean becomingActive)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;

  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  if(becomingActive)
  {
    SetPortWindowPort(windowRef);

    (*textEditStrucHdl)->viewRect.bottom = ((((*textEditStrucHdl)->viewRect.bottom -
                                            (*textEditStrucHdl)->viewRect.top) /
                                            (*textEditStrucHdl)->lineHeight) *
                                            (*textEditStrucHdl)->lineHeight) +
                                            (*textEditStrucHdl)->viewRect.top;
    (*textEditStrucHdl)->destRect.bottom = (*textEditStrucHdl)->viewRect.bottom;
    TEActivate(textEditStrucHdl);
    ActivateControl((*docStrucHdl)->vScrollbarRef);
    doAdjustScrollbar(windowRef);
    doAdjustCursor(windowRef);
  }
  else
  {
    TEDeactivate(textEditStrucHdl);
    DeactivateControl((*docStrucHdl)->vScrollbarRef);
  }
}

// ************************************************************************** doNewDocWindow

WindowRef  doNewDocWindow(void)
{
  WindowRef          windowRef;
  OSStatus           osError;
  Rect               contentRect = {  100,100,400,595 } ;
  WindowAttributes   attributes  = kWindowStandardHandlerAttribute |
                                   kWindowStandardDocumentAttributes;
  docStructureHandle docStrucHdl;
  Rect               portRect, destAndViewRect;
  EventTypeSpec    windowEvents[] = {  {  kEventClassWindow,   kEventWindowDrawContent     } ,
                                       {  kEventClassWindow,   kEventWindowActivated      } ,
                                       {  kEventClassWindow,   kEventWindowDeactivated    } ,
                                       {  kEventClassWindow,   kEventWindowClickContentRgn } ,
```

```
                                    {  kEventClassWindow,   kEventWindowClose          } ,
                                    {  kEventClassMouse,    kEventMouseDown            } ,
                                    {  kEventClassKeyboard, kEventRawKeyDown           } ,
                                    {  kEventClassKeyboard,
kEventRawKeyRepeat          }  } ;

  osError = CreateNewWindow(kDocumentWindowClass,attributes,&contentRect,&windowRef);
  if(osError != noErr)
  {
    doErrorAlert(eWindow);
    return NULL;
  }

  ChangeWindowAttributes(windowRef,0,kWindowResizableAttribute);
  RepositionWindow(windowRef,NULL,kWindowCascadeOnMainScreen);
  SetWTitle(windowRef,"\puntitled");
  SetPortWindowPort(windowRef);
  TextSize(10);

  InstallWindowEventHandler(windowRef,doGetHandlerUPP(),GetEventTypeCount(windowEvents),
                            windowEvents,0,NULL);

  if(!(docStrucHdl = (docStructureHandle) NewHandle(sizeof(docStructure))))
  {
    doErrorAlert(eDocStructure);
    return NULL;
  }
  SetWRefCon(windowRef,(SInt32) docStrucHdl);
  gNumberOfWindows ++;

  (*docStrucHdl)->vScrollbarRef = GetNewControl(rVScrollbar,windowRef);

  GetWindowPortBounds(windowRef,&portRect);
  destAndViewRect = portRect;
  destAndViewRect.right -= 15;
  destAndViewRect.bottom -= 15;
  InsetRect(&destAndViewRect,2,2);

  MoveHHi((Handle) docStrucHdl);
  HLock((Handle) docStrucHdl);

  if(!((*docStrucHdl)->textEditStrucHdl = TENew(&destAndViewRect,&destAndViewRect)))
  {
    DisposeWindow(windowRef);
    gNumberOfWindows --;
    DisposeHandle((Handle) docStrucHdl);
    doErrorAlert(eEditRecord);
    return NULL;
  }

  HUnlock((Handle) docStrucHdl);

  TESetClickLoop(gCustomClickLoopUPP,(*docStrucHdl)->textEditStrucHdl);
  TEAutoView(true,(*docStrucHdl)->textEditStrucHdl);
  TEFeatureFlag(teFOutlineHilite,1,(*docStrucHdl)->textEditStrucHdl);

  ShowWindow(windowRef);

  return windowRef;
}

// ********************************************************************** doGetHandlerUPP

EventHandlerUPP  doGetHandlerUPP(void)
{
  static EventHandlerUPP windowEventHandlerUPP;
```

```
  if(windowEventHandlerUPP == NULL)
    windowEventHandlerUPP = NewEventHandlerUPP((EventHandlerProcPtr) windowEventHandler);

  return windowEventHandlerUPP;
}

// ********************************************************************** customClickLoop

Boolean  customClickLoop(void)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  GrafPtr            oldPort;
  RgnHandle          oldClip;
  Rect               tempRect, portRect;
  Point              mouseXY;
  SInt16             linesToScroll = 0;

  windowRef = FrontWindow();
  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);
  oldClip = NewRgn();
  GetClip(oldClip);
  SetRect(&tempRect,-32767,-32767,32767,32767);
  ClipRect(&tempRect);

  GetMouse(&mouseXY);
  GetWindowPortBounds(windowRef,&portRect);

  if(mouseXY.v < portRect.top)
  {
    linesToScroll = 1;
    doSetScrollBarValue((*docStrucHdl)->vScrollbarRef,&linesToScroll);
    if(linesToScroll != 0)
      TEScroll(0,linesToScroll * ((*textEditStrucHdl)->lineHeight),textEditStrucHdl);
  }
  else if(mouseXY.v > portRect.bottom)
  {
    linesToScroll = -1;
    doSetScrollBarValue((*docStrucHdl)->vScrollbarRef,&linesToScroll);
    if(linesToScroll != 0)
      TEScroll(0,linesToScroll * ((*textEditStrucHdl)->lineHeight),textEditStrucHdl);
  }

  if(linesToScroll != 0)
    doDrawDataPanel(windowRef);

  SetClip(oldClip);
  DisposeRgn(oldClip);
  SetPort(oldPort);

  return true;
}

// ********************************************************************** doSetScrollBarValue

void  doSetScrollBarValue(ControlRef controlRef,SInt16 *linesToScroll)
{
  SInt16 controlValue, controlMax;

  controlValue = GetControlValue(controlRef);
  controlMax = GetControlMaximum(controlRef);
  *linesToScroll = controlValue - *linesToScroll;
```

```
  if(*linesToScroll < 0)
    *linesToScroll = 0;
  else if(*linesToScroll > controlMax)
    *linesToScroll = controlMax;

  SetControlValue(controlRef,*linesToScroll);
  *linesToScroll = controlValue - *linesToScroll;
}

// ********************************************************************************* doAdjustMenus

void  doAdjustMenus(void)
{
  MenuRef            fileMenuHdl, editMenuHdl;
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  ScrapRef           scrapRef;
  OSStatus           osError;
  ScrapFlavorFlags   scrapFlavorFlags;

  fileMenuHdl = GetMenuRef(mFile);
  editMenuHdl = GetMenuRef(mEdit);

  if(gNumberOfWindows > 0)
  {
    windowRef = FrontWindow();
    docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));
    textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

    EnableMenuItem(fileMenuHdl,iClose);

    if((*textEditStrucHdl)->selStart < (*textEditStrucHdl)->selEnd)
    {
      EnableMenuItem(editMenuHdl,iCut);
      EnableMenuItem(editMenuHdl,iCopy);
      EnableMenuItem(editMenuHdl,iClear);
    }
    else
    {
      DisableMenuItem(editMenuHdl,iCut);
      DisableMenuItem(editMenuHdl,iCopy);
      DisableMenuItem(editMenuHdl,iClear);
    }

    GetCurrentScrap(&scrapRef);

    osError = GetScrapFlavorFlags(scrapRef,kScrapFlavorTypeText,&scrapFlavorFlags);
    if(osError == noErr)
      EnableMenuItem(editMenuHdl,iPaste);
    else
      DisableMenuItem(editMenuHdl,iPaste);
    if((*textEditStrucHdl)->teLength > 0)
    {
      EnableMenuItem(fileMenuHdl,iSaveAs);
      EnableMenuItem(editMenuHdl,iSelectAll);
    }
    else
    {
      DisableMenuItem(fileMenuHdl,iSaveAs);
      DisableMenuItem(editMenuHdl,iSelectAll);
    }
  }
  else
  {
    DisableMenuItem(fileMenuHdl,iClose);
    DisableMenuItem(fileMenuHdl,iSaveAs);
```

```
    DisableMenuItem(editMenuHdl,iClear);
    DisableMenuItem(editMenuHdl,iSelectAll);
  }

  DrawMenuBar();
}

// ********************************************************************************** doMenuChoice

void  doMenuChoice(MenuID menuID, MenuItemIndex menuItem)
{
  if(menuID == 0)
    return;

  if(gRunningOnX)
    if(menuID == gHelpMenu)
      if(menuItem == 1)
        doHelp();

  switch(menuID)
  {
    case mAppleApplication:
      if(menuItem == iAbout)
        SysBeep(10);
      else if(menuItem == iHelp)
        doHelp();
      break;

    case mFile:
      doFileMenu(menuItem);
      break;

    case mEdit:
      doEditMenu(menuItem);
      break;
  }
}
// ************************************************************************************ doFileMenu

void  doFileMenu(MenuItemIndex menuItem)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;

  switch(menuItem)
  {
    case iNew:
      if(windowRef = doNewDocWindow())
        ShowWindow(windowRef);
      break;

    case iOpen:
      doOpenCommand();
      doAdjustScrollbar(FrontWindow());
      break;

    case iClose:
      doCloseWindow(FrontWindow());
      break;

    case iSaveAs:
      docStrucHdl = (docStructureHandle) (GetWRefCon(FrontWindow()));
      textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;
      doSaveAsFile(textEditStrucHdl);
      break;
  }
```

```
}

// ********************************************************************************** doEditMenu

void  doEditMenu(MenuItemIndex menuItem)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt32             totalSize, contigSize, newSize;
  SInt16             selectionLength;
  ScrapRef           scrapRef;
  Size               sizeOfTextData;

  windowRef = FrontWindow();

  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  switch(menuItem)
  {
    case iUndo:
      break;
    case iCut:
      if(ClearCurrentScrap() == noErr)
      {
        PurgeSpace(&totalSize,&contigSize);
        selectionLength = doGetSelectLength(textEditStrucHdl);
        if(selectionLength > contigSize)
          doErrorAlert(eNoSpaceCut);
        else
        {
          TECut(textEditStrucHdl);
          doAdjustScrollbar(windowRef);
          TEToScrap();
          if(TEToScrap() != noErr)
            ClearCurrentScrap();
        }
      }
      break;

    case iCopy:
      if(ClearCurrentScrap() == noErr)
        TECopy(textEditStrucHdl);
      TEToScrap();
      if(TEToScrap() != noErr)
        ClearCurrentScrap();
      break;

    case iPaste:
      GetCurrentScrap(&scrapRef);;
      GetScrapFlavorSize(scrapRef,kScrapFlavorTypeText,&sizeOfTextData);
      newSize = (*textEditStrucHdl)->teLength + sizeOfTextData;
      if(newSize > kMaxTELength)
        doErrorAlert(eNoSpacePaste);
      else
      {
        if(TEFromScrap() == noErr)
        {
          TEPaste(textEditStrucHdl);
          doAdjustScrollbar(windowRef);
        }
      }
      break;

    case iClear:
      TEDelete(textEditStrucHdl);
```

```
      doAdjustScrollbar(windowRef);
      break;

    case iSelectAll:
      TESetSelect(0,(*textEditStrucHdl)->teLength,textEditStrucHdl);
      break;
  }
  doDrawDataPanel(windowRef);
}

// ************************************************************************** doGetSelectLength

SInt16  doGetSelectLength(TEHandle textEditStrucHdl)
{
  SInt16 selectionLength;

  selectionLength = (*textEditStrucHdl)->selEnd - (*textEditStrucHdl)->selStart;
  return selectionLength;
}

// ************************************************************************** doAdjustScrollbar

void  doAdjustScrollbar(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt16             numberOfLines, controlMax, controlValue;

  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));;
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  numberOfLines = (*textEditStrucHdl)->nLines;
  if(*(*(*textEditStrucHdl)->hText + (*textEditStrucHdl)->teLength - 1) == kReturn)
    numberOfLines += 1;

  controlMax = numberOfLines - (((*textEditStrucHdl)->viewRect.bottom -
              (*textEditStrucHdl)->viewRect.top) /
              (*textEditStrucHdl)->lineHeight);
  if(controlMax < 0)
    controlMax = 0;
  SetControlMaximum((*docStrucHdl)-
>vScrollbarRef,controlMax);

  controlValue = ((*textEditStrucHdl)->viewRect.top - (*textEditStrucHdl)->destRect.top) /
                 (*textEditStrucHdl)->lineHeight;
  if(controlValue < 0)
    controlValue = 0;
  else if(controlValue > controlMax)
    controlValue = controlMax;

  SetControlValue((*docStrucHdl)->vScrollbarRef,controlValue);

  SetControlViewSize((*docStrucHdl)->vScrollbarRef,(*textEditStrucHdl)->viewRect.bottom -
                     (*textEditStrucHdl)->viewRect.top);

  TEScroll(0,((*textEditStrucHdl)->viewRect.top - (*textEditStrucHdl)->destRect.top) -
              (GetControlValue((*docStrucHdl)->vScrollbarRef) *
             (*textEditStrucHdl)->lineHeight),textEditStrucHdl);
}
// ************************************************************************** doAdjustCursor

void  doAdjustCursor(WindowRef windowRef)
{
  GrafPtr   oldPort;
  RgnHandle arrowRegion, iBeamRegion;
  Rect      portRect, cursorRect;
  Point     mouseXY;
```

```
  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

  arrowRegion = NewRgn();
  iBeamRegion = NewRgn();
  SetRectRgn(arrowRegion,-32768,-32768,32766,32766);

  GetWindowPortBounds(windowRef,&portRect);
  cursorRect = portRect;
  cursorRect.bottom -= 15;
  cursorRect.right  -= 15;
  LocalToGlobal(&topLeft(cursorRect));
  LocalToGlobal(&botRight(cursorRect));

  RectRgn(iBeamRegion,&cursorRect);
  DiffRgn(arrowRegion,iBeamRegion,arrowRegion);

  GetGlobalMouse(&mouseXY);

  if(PtInRgn(mouseXY,iBeamRegion))
    SetThemeCursor(kThemeIBeamCursor);
  else
    SetThemeCursor(kThemeArrowCursor);

  DisposeRgn(arrowRegion);
  DisposeRgn(iBeamRegion);

  SetPort(oldPort);
}

// ********************************************************************************** doCloseWindow

void  doCloseWindow(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;

  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));;

  DisposeControl((*docStrucHdl)->vScrollbarRef);
  TEDispose((*docStrucHdl)->textEditStrucHdl);
  DisposeHandle((Handle) docStrucHdl);
  DisposeWindow(windowRef);

  gNumberOfWindows --;
}
// ********************************************************************************** doSaveAsFile

void  doSaveAsFile(TEHandle textEditStrucHdl)
{
  OSErr            osError = noErr;
  NavDialogOptions dialogOptions;
  NavEventUPP      navEventFunctionUPP;
  WindowRef        windowRef;
  OSType           fileType;
  NavReplyRecord   navReplyStruc;
  AEKeyword        theKeyword;
  DescType         actualType;
  FSSpec           fileSpec;
  SInt16           fileRefNum;
  Size             actualSize;
  SInt32           dataLength;
  Handle           editTextHdl;

  osError = NavGetDefaultDialogOptions(&dialogOptions);

  if(osError == noErr)
```

```
  {
    windowRef = FrontWindow();

    fileType = 'TEXT';

    navEventFunctionUPP = NewNavEventUPP((NavEventProcPtr) navEventFunction);
    osError = NavPutFile(NULL,&navReplyStruc,&dialogOptions,navEventFunctionUPP,fileType,
                         'kjBb',NULL);
    DisposeNavEventUPP(navEventFunctionUPP);

    if(navReplyStruc.validRecord && osError == noErr)
    {
      if((osError = AEGetNthPtr(&(navReplyStruc.selection),1,typeFSS,&theKeyword,
                                &actualType,&fileSpec,sizeof(fileSpec),&actualSize)) == noErr)
      {

        if(!navReplyStruc.replacing)
        {
          osError = FSpCreate(&fileSpec,'kjBb',fileType,navReplyStruc.keyScript);
          if(osError != noErr)
          {
            NavDisposeReply(&navReplyStruc);
          }
        }

        if(osError == noErr)
          osError = FSpOpenDF(&fileSpec,fsRdWrPerm,&fileRefNum);

        if(osError == noErr)
        {
          SetWTitle(windowRef,fileSpec.name);
          dataLength = (*textEditStrucHdl)->teLength;
          editTextHdl = (*textEditStrucHdl)->hText;
          FSWrite(fileRefNum,&dataLength,*editTextHdl);
        }
        NavCompleteSave(&navReplyStruc,kNavTranslateInPlace);
      }

      NavDisposeReply(&navReplyStruc);
    }
  }
}

// ********************************************************************************* doOpenCommand

void  doOpenCommand(void)
{
  OSErr            osError  = noErr;
  NavDialogOptions dialogOptions;
  NavEventUPP      navEventFunctionUPP;
  NavReplyRecord   navReplyStruc;
  SInt32           index, count;
  AEKeyword        theKeyword;
  DescType         actualType;
  FSSpec           fileSpec;
  Size             actualSize;
  FInfo            fileInfo;

  osError = NavGetDefaultDialogOptions(&dialogOptions);

  if(osError == noErr)
  {
    navEventFunctionUPP = NewNavEventUPP((NavEventProcPtr) navEventFunction);
    osError = NavGetFile(NULL,&navReplyStruc,&dialogOptions,navEventFunctionUPP,NULL,NULL,
                         NULL,NULL);
    DisposeNavEventUPP(navEventFunctionUPP);
```

```
    if(osError == noErr && navReplyStruc.validRecord)
    {
      osError = AECountItems(&(navReplyStruc.selection),&count);
      if(osError == noErr)
      {
        for(index=1;index<=count;index++)
        {
          osError = AEGetNthPtr(&(navReplyStruc.selection),index,typeFSS,&theKeyword,
                                &actualType,&fileSpec,sizeof(fileSpec),&actualSize);
          {
            if((osError = FSpGetFInfo(&fileSpec,&fileInfo)) == noErr)
              doOpenFile(fileSpec);
          }
        }
      }

      NavDisposeReply(&navReplyStruc);
    }
  }
}
// ********************************************************************************** doOpenFile

void  doOpenFile(FSSpec fileSpec)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt16             fileRefNum;
  SInt32             textLength;
  Handle             textBuffer;

  if((windowRef = doNewDocWindow()) == NULL)
    return;

  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));;
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  SetWTitle(windowRef,fileSpec.name);

  FSpOpenDF(&fileSpec,fsCurPerm,&fileRefNum);

  SetFPos(fileRefNum,fsFromStart,0);
  GetEOF(fileRefNum,&textLength);

  if(textLength > 32767)
    textLength = 32767;

  textBuffer = NewHandle((Size) textLength);

  FSRead(fileRefNum,&textLength,*textBuffer);

  MoveHHi(textBuffer);
  HLock(textBuffer);

  TESetText(*textBuffer,textLength,textEditStrucHdl);

  HUnlock(textBuffer);
  DisposeHandle(textBuffer);

  FSClose(fileRefNum);

  (*textEditStrucHdl)->selStart = 0;
  (*textEditStrucHdl)->selEnd = 0;

  doDrawContent(windowRef);
}
```

```
// ************************************************************************** doDrawDataPanel

void  doDrawDataPanel(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  RGBColor           whiteColor = {  0xFFFF, 0xFFFF, 0xFFFF } ;
  RGBColor           blackColor = {  0x0000, 0x0000, 0x0000 } ;
  RGBColor           blueColor = {  0x1818, 0x4B4B, 0x8181 } ;
  ControlRef         controlRef;
  Rect               panelRect;
  Str255             textString;

  SetPortWindowPort(windowRef);

  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));;
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;
  controlRef = (*docStrucHdl)->vScrollbarRef;

  MoveTo(0,282);
  LineTo(495,282);

  RGBForeColor(&whiteColor);
  RGBBackColor(&blueColor);
  SetRect(&panelRect,0,283,495,300);
  EraseRect(&panelRect);

  MoveTo(3,295);
  DrawString("\pteLength               nLines          lineHeight");

  MoveTo(225,295);
  DrawString("\pdestRect.top              controlValue           contrlMax");

  SetRect(&panelRect,47,284,88,299);
  EraseRect(&panelRect);
  SetRect(&panelRect,124,284,149,299);
  EraseRect(&panelRect);
  SetRect(&panelRect,204,284,222,299);
  EraseRect(&panelRect);
  SetRect(&panelRect,286,284,323,299);
  EraseRect(&panelRect);
  SetRect(&panelRect,389,284,416,299);
  EraseRect(&panelRect);
  SetRect(&panelRect,472,284,495,299);
  EraseRect(&panelRect);

  NumToString((SInt32) (*textEditStrucHdl)->teLength,textString);
  MoveTo(47,295);
  DrawString(textString);

  NumToString((SInt32) (*textEditStrucHdl)->nLines,textString);
  MoveTo(124,295);
  DrawString(textString);

  NumToString((SInt32) (*textEditStrucHdl)->lineHeight,textString);
  MoveTo(204,295);
  DrawString(textString);

  NumToString((SInt32) (*textEditStrucHdl)->destRect.top,textString);
  MoveTo(286,295);
  DrawString(textString);
  NumToString((SInt32) GetControlValue(controlRef),textString);
  MoveTo(389,295);
  DrawString(textString);

  NumToString((SInt32) GetControlMaximum(controlRef),textString);
  MoveTo(472,295);
```

```
  DrawString(textString);

  RGBForeColor(&blackColor);
  RGBBackColor(&whiteColor);
}

// ********************************************************************************* doErrorAlert

void  doErrorAlert(SInt16 errorCode)
{
  Str255 errorString;
  SInt16 itemHit;

  GetIndString(errorString,rErrorStrings,errorCode);

  if(errorCode < eWindow)
  {
    StandardAlert(kAlertStopAlert,errorString,NULL,NULL,&itemHit);
    ExitToShell();
  }
  else
  {
    StandardAlert(kAlertCautionAlert,errorString,NULL,NULL,&itemHit);
  }
}

// ********************************************************************** navEventFunction

void  navEventFunction(NavEventCallbackMessage callBackSelector,NavCBRecPtr callBackParms,
                       NavCallBackUserData callBackUD)
{
}

// ********************************************************************************************
// HelpDialog.c
// ********************************************************************************************

// ..................................................................................... includes

#include <Carbon.h>

// ..................................................................................... defines

#define eHelpDialog           8
#define eHelpDocStructure     9
#define eHelpText             10
#define eHelpPicture          11
#define eHelpControls         12
#define rTextIntroduction     128
#define rTextCreatingText     129
#define rTextModifyHelp       130
#define rPictIntroductionBase 128
#define rPictCreatingTextBase 129
#define kTextInset            4

// ..................................................................................... typedefs

typedef struct
{
  Rect      bounds;
  PicHandle pictureHdl;
}  pictInfoStructure;

typedef struct
{
  TEHandle          textEditStrucHdl;
  ControlRef        scrollbarHdl;
```

```
  SInt16            pictCount;
  pictInfoStructure *pictInfoStructurePtr;
}   docStructure, ** docStructureHandle;

typedef struct
{
  RGBColor     backColor;
  PixPatHandle backPixelPattern;
  Pattern      backBitPattern;
}  backColorPattern;

// ..................................................................... global
variables

GrafPtr                gOldPort;
EventHandlerUPP        helpWindowEventHandlerUPP;
ControlUserPaneDrawUPP userPaneDrawFunctionUPP;
ControlActionUPP       actionFunctionUPP;
SInt16                 gTextResourceID;
SInt16                 gPictResourceBaseID;
RgnHandle              gSavedClipRgn  = NULL;

// ..................................................................... function
prototypes

void      doHelp                (void);
OSStatus  helpWindowEventHandler (EventHandlerCallRef,EventRef,void *);
void      userPaneDrawFunction   (ControlRef,SInt16);
Boolean   doGetText              (WindowRef,SInt16,Rect);
Boolean   doGetPictureInfo       (WindowRef,SInt16);
void      actionFunction         (ControlRef,SInt16);
void      doScrollTextAndPicts   (WindowRef);
void      doDrawPictures         (WindowRef,Rect *);
void      doCloseHelp            (WindowRef);
void      doDisposeDescriptors   (void);
void      doSetBackgroundWhite   (void);

extern void  doErrorAlert        (SInt16);
// ********************************************************************************* doHelp

void  doHelp(void)
{
  OSStatus           osError;
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  ControlRef         controlRef;
  ControlID          controlID;
  Rect               windowRect      = {  0,  0,  353,382 } ;
  Rect               pushButtonRect  = {  312,297,332,366 } ;
  Rect               userPaneRect    = {  16, 16, 296,351 } ;
  Rect               scrollBarRect   = {  16, 350,296,366 } ;
  Rect               popupButtonRect = {  312,12, 332,256 } ;
  Rect               destRect, viewRect;
  EventTypeSpec      dialogEvents[] = { {  kEventClassControl, kEventControlClick }  } ;

  GetPort(&gOldPort);

  // ..................................................... create universal procedure
pointers

  helpWindowEventHandlerUPP = NewEventHandlerUPP((EventHandlerProcPtr)
                                                 helpWindowEventHandler);
  userPaneDrawFunctionUPP = NewControlUserPaneDrawUPP((ControlUserPaneDrawProcPtr)
                                                      userPaneDrawFunction);
  actionFunctionUPP = NewControlActionUPP((ControlActionProcPtr) actionFunction);

  // ................................................. create modal class window
```

```
osError = CreateNewWindow(kMovableModalWindowClass,kWindowStandardHandlerAttribute,
                          &windowRect,&windowRef);
if(osError == noErr)
{
  RepositionWindow(windowRef,FrontWindow(),kWindowAlertPositionOnMainScreen);
  SetThemeWindowBackground(windowRef,kThemeBrushDialogBackgroundActive,false);

  InstallWindowEventHandler(windowRef,helpWindowEventHandlerUPP,
                            GetEventTypeCount(dialogEvents),dialogEvents,windowRef,NULL);

  if(!(docStrucHdl = (docStructureHandle) NewHandle(sizeof(docStructure))))
  {
    doErrorAlert(eHelpDocStructure);
    DisposeWindow(windowRef);
    doDisposeDescriptors();
    return;
  }

  SetWRefCon(windowRef,(SInt32) docStrucHdl);
  SetPortWindowPort(windowRef);

  // ......................... create root, push button, user pane, and scroll bar controls

  CreateRootControl(windowRef,&controlRef);
  if((osError = CreatePushButtonControl(windowRef,&pushButtonRect,CFSTR("OK"),&controlRef))
      == noErr)
  {
    SetWindowDefaultButton(windowRef,controlRef);
    controlID.id = 'done';
    SetControlID(controlRef,&controlID);
  }

  if(osError == noErr)
  {
    if((osError = CreateUserPaneControl(windowRef,&userPaneRect,0,&controlRef)) == noErr)
    {
      SetControlData(controlRef,kControlEntireControl,kControlUserPaneDrawProcTag,
                     sizeof(userPaneDrawFunctionUPP),(Ptr) &userPaneDrawFunctionUPP);
    }
  }

  if(osError == noErr)
  {
    if((osError = CreateScrollBarControl(windowRef,&scrollBarRect,0,0,1,0,true,
                                         actionFunctionUPP,&controlRef)) == noErr)
    (*docStrucHdl)->scrollbarHdl = controlRef;
    controlID.id = 'scro';
    SetControlID(controlRef,&controlID);
  }

  if(osError == noErr)
  {
    if((osError = CreatePopupButtonControl(windowRef,&popupButtonRect,CFSTR("Title:"),131,
                                           false,-1,0,0,&controlRef)) == noErr)
    controlID.id = 'popu';
    SetControlID(controlRef,&controlID);
  }

  if(osError != noErr)
  {
    doErrorAlert(eHelpControls);
    DisposeWindow(windowRef);
    doDisposeDescriptors();
    return;
  }
}
```

```
  else
  {
    doErrorAlert(eHelpDialog);
    doDisposeDescriptors();
    return;
  }
  // ........................... set destination and view rectangles, create TextEdit
structure

  InsetRect(&userPaneRect,kTextInset,kTextInset / 2);
  destRect = viewRect = userPaneRect;
    (*docStrucHdl)->textEditStrucHdl = TEStyleNew(&destRect,&viewRect);
  // ..................... initialise picture information structure field of document structure

  (*docStrucHdl)->pictInfoStructurePtr = NULL;

  // ........................... assign resource IDs of first topic's 'TEXT'/'styl' resources

  gTextResourceID      = rTextIntroduction;
  gPictResourceBaseID  = rPictIntroductionBase;

  // ..................................... load text resources and insert into edit structure

  if(!(doGetText(windowRef,gTextResourceID,viewRect)))
  {
    doCloseHelp(windowRef);
    doDisposeDescriptors();
    return;
  }

  // ......... search for option-space charas in text and load same number of 'PICT' resources

  if(!(doGetPictureInfo(windowRef,gPictResourceBaseID)))
  {
    doCloseHelp(windowRef);
    doDisposeDescriptors();
    return;
  }

  // ............................... create an empty region for saving the old clipping region

  gSavedClipRgn = NewRgn();

  // ............................................................ show window and run modal
loop

  ShowWindow(windowRef);
  RunAppModalLoopForWindow(windowRef);
}

// ********************************************************************** helpWindowEventHandler

OSStatus  helpWindowEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                                 void *userData)
{
  OSStatus           result = eventNotHandledErr;
  WindowRef          windowRef;
  UInt32             eventClass;
  UInt32             eventKind;
  Point              mouseLocation;
  ControlRef         controlRef;
  ControlPartCode    controlPartCode;
  ControlID          controlID;
  MenuItemIndex      menuItem;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  Rect               viewRect;
```

```
windowRef  = userData;
eventClass = GetEventClass(eventRef);
eventKind  = GetEventKind(eventRef);

if(eventClass == kEventClassControl)
{
  if(eventKind == kEventControlClick)
  {
    GetEventParameter(eventRef,kEventParamMouseLocation,typeQDPoint,NULL,
                      sizeof(mouseLocation),NULL,&mouseLocation);
    GlobalToLocal(&mouseLocation);
    controlRef = FindControlUnderMouse(mouseLocation,windowRef,&controlPartCode);
    if(controlRef)
    {
      GetControlID(controlRef,&controlID);
      if(controlID.id == 'done')                                          // push button
      {
        if(TrackControl(controlRef,mouseLocation,NULL))
        {
          QuitAppModalLoopForWindow(windowRef);
          doCloseHelp(windowRef);
          doDisposeDescriptors();
          result = noErr;
        }
      }
      if(controlID.id == 'scro')                                           // scroll bar
      {
        TrackControl(controlRef,mouseLocation,actionFunctionUPP);
        result = noErr;
      }
      else if(controlID.id == 'popu')                                  // pop-up menu button
      {
        TrackControl(controlRef,mouseLocation,(ControlActionUPP) -1);
        menuItem = GetControlValue(controlRef);
        switch(menuItem)
        {
          case 1:
            gTextResourceID     = rTextIntroduction;
            gPictResourceBaseID = rPictIntroductionBase;
            break;

          case 2:
            gTextResourceID     = rTextCreatingText;
            gPictResourceBaseID = rPictCreatingTextBase;
            break;

          case 3:
            gTextResourceID     = rTextModifyHelp;
            break;
        }

        docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
        textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;
        viewRect = (*textEditStrucHdl)->viewRect;
        if(!(doGetText(windowRef,gTextResourceID,viewRect)))
        {
          doCloseHelp(windowRef);
          doDisposeDescriptors();
          return;
        }

        if(!(doGetPictureInfo(windowRef,gPictResourceBaseID)))
        {
          doCloseHelp(windowRef);
          doDisposeDescriptors();
          return;
        }
```

```
          doDrawPictures(windowRef,&viewRect);

          result = noErr;
        }
      }
    }
  }

  return result;
}

// ********************************************************************** userPaneDrawFunction

void  userPaneDrawFunction(ControlRef controlRef,SInt16 thePart)
{
  Rect               itemRect, viewRect;
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  Boolean           inState;

  windowRef = GetControlOwner(controlRef);

  GetControlBounds(controlRef,&itemRect);
  InsetRect(&itemRect,1,1);
  itemRect.right += 15;

  if(IsWindowVisible(windowRef))
    inState = IsWindowHilited(windowRef);
  DrawThemeListBoxFrame(&itemRect,inState);

  doSetBackgroundWhite();
  EraseRect(&itemRect);

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;
  viewRect = (*textEditStrucHdl)->viewRect;

  TEUpdate(&viewRect,textEditStrucHdl);
  doDrawPictures(windowRef,&viewRect);
}
// ********************************************************************************** doGetText

Boolean  doGetText(WindowRef windowRef,SInt16 textResourceID,Rect viewRect)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  Handle             helpTextHdl;
  StScrpHandle       stylScrpStrucHdl;
  SInt16             numberOfLines, heightOfText, heightToScroll;

  doSetBackgroundWhite();

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  TESetSelect(0,32767,textEditStrucHdl);
  TEDelete(textEditStrucHdl);

  (*textEditStrucHdl)->destRect = (*textEditStrucHdl)->viewRect;
  SetControlValue((*docStrucHdl)->scrollbarHdl,0);

  helpTextHdl = GetResource('TEXT',textResourceID);
  if(helpTextHdl == NULL)
  {
    doErrorAlert(eHelpText);
```

```
    return false;
  }

  stylScrpStrucHdl = (StScrpHandle) GetResource('styl',textResourceID);
  if(stylScrpStrucHdl == NULL)
  {
    doErrorAlert(eHelpText);
    return false;
  }

  TEStyleInsert(*helpTextHdl,GetHandleSize(helpTextHdl),stylScrpStrucHdl,textEditStrucHdl);

  ReleaseResource(helpTextHdl);
  ReleaseResource((Handle) stylScrpStrucHdl);

  numberOfLines = (*textEditStrucHdl)->nLines;
  heightOfText = TEGetHeight((SInt32) numberOfLines,1,textEditStrucHdl);

  if(heightOfText > (viewRect.bottom - viewRect.top))
  {
    heightToScroll = TEGetHeight((SInt32) numberOfLines,1,textEditStrucHdl) -
                              (viewRect.bottom - viewRect.top);
    SetControlMaximum((*docStrucHdl)->scrollbarHdl,heightToScroll);
    ActivateControl((*docStrucHdl)->scrollbarHdl);
    SetControlViewSize((*docStrucHdl)->scrollbarHdl,(*textEditStrucHdl)->viewRect.bottom -
                       (*textEditStrucHdl)->viewRect.top);
  }
  else
  {
    DeactivateControl((*docStrucHdl)->scrollbarHdl);
  }
  return true;
}

// ********************************************************************** doGetPictureInfo

Boolean  doGetPictureInfo(WindowRef windowRef,SInt16 firstPictID)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  Handle             textHdl;
  SInt32             offset, textSize;
  SInt16             numberOfPicts, a, lineHeight, fontAscent;
  SInt8              optionSpace[1] = "\xCA";
  pictInfoStructure  *pictInfoPtr;
  Point              picturePoint;
  TextStyle          whatStyle;

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

  if((*docStrucHdl)->pictInfoStructurePtr != NULL)
  {
    for(a=0;a<(*docStrucHdl)->pictCount;a++)
      ReleaseResource((Handle) (*docStrucHdl)->pictInfoStructurePtr[a].pictureHdl);

    DisposePtr((Ptr) (*docStrucHdl)->pictInfoStructurePtr);
    (*docStrucHdl)->pictInfoStructurePtr = NULL;
  }

  (*docStrucHdl)->pictCount = 0;

  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;
  textHdl = (*textEditStrucHdl)->hText;

  textSize = GetHandleSize(textHdl);
  offset = 0;
  numberOfPicts = 0;
```

```
  HLock(textHdl);

  offset = Munger(textHdl,offset,optionSpace,1,NULL,0);
  while((offset >= 0) && (offset <= textSize))
  {
    numberOfPicts++;
    offset++;
    offset = Munger(textHdl,offset,optionSpace,1,NULL,0);
  }

  if(numberOfPicts == 0)
  {
    HUnlock(textHdl);
    return true;
  }

  pictInfoPtr = (pictInfoStructure *) NewPtr(sizeof(pictInfoStructure) * numberOfPicts);
  (*docStrucHdl)->pictInfoStructurePtr = pictInfoPtr;
  offset = 0L;

  for(a=0;a<numberOfPicts;a++)
  {
    pictInfoPtr[a].pictureHdl = GetPicture(firstPictID + a);
    if(pictInfoPtr[a].pictureHdl == NULL)
    {
      doErrorAlert(eHelpPicture);
      return false;
    }

    offset = Munger(textHdl,offset,optionSpace,1,NULL,0);
    picturePoint = TEGetPoint((SInt16)offset,textEditStrucHdl);

    TEGetStyle(offset,&whatStyle,&lineHeight,&fontAscent,textEditStrucHdl);
    picturePoint.v -= lineHeight;
    offset++;
    pictInfoPtr[a].bounds = (**pictInfoPtr[a].pictureHdl).picFrame;

    OffsetRect(&pictInfoPtr[a].bounds,
               (((*textEditStrucHdl)->destRect.right + (*textEditStrucHdl)->destRect.left) -
               (pictInfoPtr[a].bounds.right + pictInfoPtr[a].bounds.left) ) / 2,
               - pictInfoPtr[a].bounds.top + picturePoint.v);
  }

  (*docStrucHdl)->pictCount = a;

  HUnlock(textHdl);

  return true;
}

// ********************************************************************** actionFunction

void  actionFunction(ControlRef scrollbarHdl,SInt16 partCode)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt16             delta, oldValue, offset, lineHeight, fontAscent;
  Point              thePoint;
  Rect               viewRect, portRect;
  TextStyle          style;

  if(partCode)
  {
    windowRef = GetControlOwner(scrollbarHdl);
    docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
```

```
textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;
viewRect = (*textEditStrucHdl)->viewRect;
thePoint.h = viewRect.left + kTextInset;

if(partCode != kControlIndicatorPart)
{
  switch(partCode)
  {
    case kControlUpButtonPart:
      thePoint.v = viewRect.top - 4;
      offset = TEGetOffset(thePoint,textEditStrucHdl);
      thePoint = TEGetPoint(offset,textEditStrucHdl);
      TEGetStyle(offset,&style,&lineHeight,&fontAscent,textEditStrucHdl);
      delta = thePoint.v - lineHeight - viewRect.top;
      break;

    case kControlDownButtonPart:
      thePoint.v = viewRect.bottom + 2;
      offset = TEGetOffset(thePoint,textEditStrucHdl);
      thePoint = TEGetPoint(offset,textEditStrucHdl);
      delta = thePoint.v - viewRect.bottom;
      break;

    case kControlPageUpPart:
      thePoint.v = viewRect.top + 2;
      offset = TEGetOffset(thePoint,textEditStrucHdl);
      thePoint = TEGetPoint(offset,textEditStrucHdl);
      TEGetStyle(offset,&style,&lineHeight,&fontAscent,textEditStrucHdl);
      thePoint.v += lineHeight - fontAscent;
      thePoint.v -= viewRect.bottom - viewRect.top;
      offset = TEGetOffset(thePoint,textEditStrucHdl);
      thePoint = TEGetPoint(offset,textEditStrucHdl);
      TEGetStyle(offset,&style,&lineHeight,&fontAscent,textEditStrucHdl);
      delta = thePoint.v - viewRect.top;
      if(offset == 0)
        delta -= lineHeight;
      break;

    case kControlPageDownPart:
      thePoint.v = viewRect.bottom - 2;
      offset = TEGetOffset(thePoint,textEditStrucHdl);
      thePoint = TEGetPoint(offset,textEditStrucHdl);
      TEGetStyle(offset,&style,&lineHeight,&fontAscent,textEditStrucHdl);
      thePoint.v -= fontAscent;
      thePoint.v += viewRect.bottom - viewRect.top;
      offset = TEGetOffset(thePoint,textEditStrucHdl);
      thePoint = TEGetPoint(offset,textEditStrucHdl);
      TEGetStyle(offset,&style,&lineHeight,&fontAscent,textEditStrucHdl);
      delta =  thePoint.v - lineHeight - viewRect.bottom;
      if(offset == (**textEditStrucHdl).teLength)
        delta += lineHeight;
      break;
  }

  oldValue = GetControlValue(scrollbarHdl);

  if(((delta < 0) && (oldValue > 0)) || ((delta > 0) &&
     (oldValue < GetControlMaximum(scrollbarHdl))))
  {
    GetClip(gSavedClipRgn);
    GetWindowPortBounds(windowRef,&portRect);
    ClipRect(&portRect);
    SetControlValue(scrollbarHdl,oldValue + delta);
    SetClip(gSavedClipRgn);
  }
}
```

```
    doScrollTextAndPicts(windowRef);
  }
}

// ************************************************************************ doScrollTextAndPicts

void  doScrollTextAndPicts(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt16             scrollDistance, oldScroll;
  Rect               updateRect;

  doSetBackgroundWhite();

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  oldScroll = (*textEditStrucHdl)->viewRect.top -(*textEditStrucHdl)->destRect.top;
  scrollDistance = oldScroll - GetControlValue((*docStrucHdl)->scrollbarHdl);
  if(scrollDistance == 0)
    return;

  TEScroll(0,scrollDistance,textEditStrucHdl);

  if((*docStrucHdl)->pictCount == 0)
    return;

  updateRect = (*textEditStrucHdl)->viewRect;

  if(scrollDistance > 0)
  {
    if(scrollDistance < (updateRect.bottom - updateRect.top))
      updateRect.bottom = updateRect.top + scrollDistance;
  }
  else
  {
    if( - scrollDistance < (updateRect.bottom - updateRect.top))
      updateRect.top = updateRect.bottom + scrollDistance;
  }

  doDrawPictures(windowRef,&updateRect);
}
// ************************************************************************** doDrawPictures

void  doDrawPictures(WindowRef windowRef,Rect *updateRect)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt16             pictCount, pictIndex, vOffset;
  PicHandle          thePictHdl;
  Rect               pictLocRect, dummyRect;

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  vOffset = (*textEditStrucHdl)->destRect.top -
            (*textEditStrucHdl)->viewRect.top - kTextInset;
  pictCount = (*docStrucHdl)->pictCount;

  for(pictIndex = 0;pictIndex < pictCount;pictIndex++)
  {
    pictLocRect = (*docStrucHdl)->pictInfoStructurePtr[pictIndex].bounds;
    OffsetRect(&pictLocRect,0,vOffset);

    if(!SectRect(&pictLocRect,updateRect,&dummyRect))
      continue;
```

```
    thePictHdl = (*docStrucHdl)->pictInfoStructurePtr[pictIndex].pictureHdl;

    LoadResource((Handle) thePictHdl);
    HLock((Handle) thePictHdl);

    GetClip(gSavedClipRgn);
    ClipRect(updateRect);
    DrawPicture(thePictHdl,&pictLocRect);

    SetClip(gSavedClipRgn);
    HUnlock((Handle) thePictHdl);
  }
}

// ********************************************************************************** doCloseHelp

void  doCloseHelp(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt16             a;

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  if(gSavedClipRgn)
    DisposeRgn(gSavedClipRgn);
  if((*docStrucHdl)->textEditStrucHdl)
    TEDispose((*docStrucHdl)->textEditStrucHdl);

  if((*docStrucHdl)->pictInfoStructurePtr)
  {
    for(a=0;a<(*docStrucHdl)->pictCount;a++)
      ReleaseResource((Handle) (*docStrucHdl)->pictInfoStructurePtr[a].pictureHdl);
    DisposePtr((Ptr) (*docStrucHdl)->pictInfoStructurePtr);
  }

  DisposeHandle((Handle) docStrucHdl);
  DisposeWindow(windowRef);
  SetPort(gOldPort);
}

// ********************************************************************** doDisposeDescriptors

void  doDisposeDescriptors(void)
{
  DisposeEventHandlerUPP(helpWindowEventHandlerUPP);
  DisposeControlUserPaneDrawUPP(userPaneDrawFunctionUPP);
  DisposeControlActionUPP(actionFunctionUPP);
}

// ************************************************************************ doSetBackgroundWhite

void  doSetBackgroundWhite(void)
{
  RGBColor whiteColor = {  0xFFFF, 0xFFFF, 0xFFFF } ;
  Pattern  whitePattern;

  RGBBackColor(&whiteColor);
  BackPat(GetQDGlobalsWhite(&whitePattern));
}

// ************************************************************************************************
```

Demonstration Program MonoTextEdit Comments

When this program is run, the user should explore both the text editor and the Help dialog.

Text Editor

In the text editor, the user should perform all the actions usually associated with a simple text editor, that is

- Open a new document window, open an existing 'TEXT' file for display in a new document window, and save a document to a 'TEXT' file. (A 'TEXT' file titled "MonoTextEdit Document" is included.)

- Enter new text and use the Edit menu Cut, Copy, Paste, and Clear commands to edit the text. (Pasting between documents and other applications is supported.)

- Select text by clicking and dragging, double-clicking a word, shift-clicking, and choosing the Select All command from the Edit menu. Also select large amounts of text by clicking in the text and dragging the cursor above or below the window so as to invoke auto-scrolling.

- Scroll a large document by dragging the scroll box/scroller (live scrolling is used), clicking once in a scroll arrow or gray area/track, and holding the mouse down in a scroll arrow or gray area/track.

Whenever any action is taken, the user should observe the changes to the values displayed in the data panel at the bottom of each window. In particular, the relationship between the destination rectangle and scroll bar control value should be noted.

The user should also note that outline highlighting is activated for all windows and that the forward-delete key is supported by the application. (The forward-delete key is not supported by TextEdit.)

Help Dialog

The user should choose MonoTextEdit Help from the Mac OS 8/9 Apple menu or Mac OS X Help menu to open the Help dialog and then scroll through the three help topics, which may be chosen in the pop-up menu at the bottom of the dialog. The help topics contain documentation on the Help dialog that supplements the source code comments below.

MonoTextEdit.c

defines

`kMaxTELength` represents the maximum allowable number of bytes in a TextEdit structure. `kTab`, `kBackSpace`, `kForwardDelete`, `kReturn`, and `kEscape` representing the character codes generated by the tab, delete, forward delete, Return and escape keys.

typedefs

The `docStructure` data type will be used for a small document structure comprising a handle to a TextEdit structure and a handle to a vertical scroll bar.

Global Variables

`scrollActionFunctionUPP` will be assigned a universal procedure pointer to an action (callback) function for the scroll bar. `customClickLoopUPP` will be assigned a universal procedure pointer to a custom click loop (callback) function. `gOldControlValue` will be assigned the scroll bar's control value.

main

The `main` function creates universal procedure pointers for the application-defined scroll action and custom click loop (callback) functions and installs a timer set to fire repeatedly at the interval returned by a call to `GetCaretTime`. When the timer fires, the function `doIdle` is called.

windowEventHandler

When the `kEventClassMouse` event type is received, if the call to `FindControlUnderMouse` reveals that a control (that is, the vertical scroll bar) is under the mouse, `TrackControl` is called with a universal procedure pointer to an action function passed in the third parameter.

When the `kEventRawKeyDown` and `kEventRawKeyRepeat` event types are received, the function `doKeyEvent` is called only if the Command key was not down.

doIdle

`doIdle` is called whenever the installed timer fires.

The first line gets a reference to the front window. If the front window is a document

window, a handle to the window's document structure is retrieved. A handle to the TextEdit structure associated with the window is stored in the document structure's `textEditStrucHdl` field. This is passed in the call to `TEIdle`, which blinks the insertion point caret.

doKeyEvent

`doKeyEvent` is called when the `kEventRawKeyDown` and `kEventRawKeyRepeat` event types are received. It handles all key-down events that are not Command key equivalents.

If the character code is equal to that for the escape key or lower, and except if it is the carriage return or backspace character, the function simply returns.

The first three lines get a handle to the TextEdit structure whose handle is stored in the front window's document structure.

The next line filters out the tab key character code. (TextEdit does not support the tab key and some applications may need to provide a tab key handler.)

The next character code to be filtered out is the forward-delete key character code. TextEdit does not recognize this key, so this else if block provides forward-delete key support for the program. The first line in this block gets the current selection length from the TextEdit structure. If this is zero (that is, there is no selection range and an insertion point is being displayed), the `selEnd` field is incremented by one. This, in effect, creates a selection range comprising the character following the insertion point. TEDelete deletes the current selection range from the TextEdit structure. Such deletions could change the number of text lines in the TextEdit structure, requiring the vertical scroll bar to be adjusted; hence the call to the function `doAdjustScrollbar`.

Processing of those character codes that have not been filtered out is performed in the else block. A new character must not be allowed to be inserted if the TextEdit limit of 32,767 characters will be exceeded. Accordingly, and given that TEKey replaces the selection range with the character passed to it, the first step is to get the current selection length. If the current number of characters minus the selection length plus 1 is less than 32,767, the character code is passed to TEKey for insertion into the TextEdit structure. In addition, and because all this could change the number of lines in the TextEdit structure, the scroll bar adjustment function is called.

If the TextEdit limit will be exceeded by accepting the character, an alert is invoked advising the user of the situation.

The last line calls a function that prints data extracted from the edit text and control structures at the bottom of the window.

scrollActionFunction

`scrollActionFunction` is associated with the vertical scroll bar. It is the callback function which will be repeatedly called by `TrackControl` when the `kEventMouseDown` event type is received. It will be called repeatedly while the mouse button remains down in the scroll box/scroller, scroll arrows or gray areas/track of the vertical scroll bar.

The first line gets a reference to the window object for the window which "owns" the control. The next two lines get a handle to the TextEdit structure associated with the window.

Within the outer if block, the first if block executes if the control part is not the scroll box/ scroller (that is, the indicator). The purpose of the switch is to get a value into the variable `linesToScroll`. If the mouse-down was in a scroll arrow, that value will be 1. If the mouse-down was in a gray area/track, that value will be equivalent to one less than the number of text lines that will fit in the view rectangle. (Subtracting 1 from the total number of lines that will fit in the view rectangle ensures that the line of text at the bottom/top of the view rectangle prior to a gray area/track scroll will be visible at the top/bottom of the window after the scroll.)

Immediately after the switch, the value in `linesToScroll` is changed to a negative value if the mouse-down occurred in either the down scroll arrow or down gray area/track.

The next block ensures that no scrolling action will occur if the document is currently scrolled fully up (control value equals control maximum) or fully down (control value equals 0). In either case, `linesToScroll` will be set to 0, meaning that the call to `TEScroll` near the end of the function will not occur.

`SetControlValue` sets the control value to the value just previously calculated, that is, to the current control value minus the value in `linesToScroll`.

The next line sets the value in `linesToScroll` back to what it was before the line `linesToScroll = controlvalue - linesToScroll` executed. This value, multiplied by the value in the `lineHeight` field of the TextEdit structure, is later passed to `TEScroll` as the parameter which specifies the number of pixels to scroll.

If the control part is the scroll box/scroller (that is, the indicator), the variable `linesToScroll` is assigned a value equal to the control's value as it was the last time this function was called minus the control's current value. The global variable which holds the control's "old" value is then assigned the control's current value preparatory to the next call to this function.

With the number of lines to scroll determined, `TEScroll` is called to scroll the text within

the view rectangle by the number of pixels specified in the second parameter. (Positive values scroll the text towards the bottom of the screen. Negative values scroll the text towards the top.)

The last line is for demonstration purposes only. It calls the function that prints data extracted from the edit and control structures at the bottom of the window.

doInContent

`doInContent` is called when the `kEventWindowClickContentRgn` event type is received.

The first three lines retrieve a handle to the TextEdit structure associated with the front window. The call to `PtInRect` checks whether the mouse-down occurred within the view rectangle. (Note that the view rectangle is in local coordinates, so the mouse-down coordinates passed as the first parameter to the `PtInRect` call must also be in local coordinates.) If the mouse-down was in the view rectangle, `TEClick` is called to advise TextEdit of the mouse-down event. Note that the position of the shift key is passed in the second parameter. (`TEClick`'s behavior depends on the position of the shift key.)

doDrawContent

`doDrawContent` is called when the `kEventWindowDrawContent` event type is received.

The first two lines get the handle to the TextEdit structure associated with the window.

`UpdateControls` is called to draw the scroll bar. The call to `TEUpdate` draws the text currently in the TextEdit structure.

doActivateDeactivate

`doActivateDeactivatew` performs window activation/deactivation. It is called when the `kEventWindowActivated` and `kEventWindowDecativated` event types are received.

The first two lines retrieve a handle to the TextEdit structure for the window. If the window is becoming active, its graphics port is set as the current graphics port. The bottom of the view rectangle is then adjusted so that the height of the view rectangle is an exact multiple of the value in the `lineHeight` field of the TextEdit structure. (This avoids the possibility of only part of the full height of a line of text appearing at the bottom of the view rectangle.) `TEActivate` activates the TextEdit structure associated with the window, `ActivateControl` activates the scroll bar, `doAdjustScrollbar` adjusts the scroll bar, and `doAdjustCursor` adjusts the cursor shape.

If the window is becoming inactive, `TEDeactivate` deactivates the TextEdit structure associated with the window and `DeactivateControl` deactivates the scroll bar.

doNewDocWindow

`doNewDocWindow` is called at program launch and when the user chooses New or Open from the File menu. It opens a new window, associates a document structure with that window, creates a vertical scroll bar, creates a monostyled TextEdit structure, installs the custom click loop (callback) function, enables automatic scrolling, and enables outline highlighting.

The call to `CreateNewWindow` and the following block creates a new window with the standard document window attributes less the size box/resize control. Note that the window's graphics port is set as the current port before the later call to `TENew`. (Because the TextEdit structure assumes the drawing environment specified in the graphics port structure, setting the graphics port must be done before `TENew` creates the TextEdit structure.)

The call to `TextSize` sets the text size. This, together with the default application font, will be copied from the graphics port to the TextEdit structure when TENew is called.

After the window event handler is installed, a document structure is created and the handle stored in the window's window object. The following line increments the global variable that keeps track of the number of open windows. `GetNewControl` creates a vertical scroll bar and assigns a handle to it to the appropriate field of the document structure. The next block establishes the view and destination rectangles two pixels inside the window's port rectangle less the scroll bar.

`MoveHHi` and `HLock` move the document structure high and lock it. A monostyled TextEdit structure is then created by `TENew` and its handle is assigned to the appropriate field of the document structure. (If this call is not successful, the window and scroll bar are disposed of, an error alert is displayed, and the function returns.) The handle to the document structure is then unlocked.

`TESetClickLoop` installs the universal procedure pointer to the custom click loop (callback) function `customClickLoop` in the `clickLoop` field of the TextEdit structure. `TEAutoView` enables automatic scrolling for the TextEdit structure. `TEFeatureFlag` enables outline highlighting for the TextEdit structure.

The last line returns a reference to the newly opened window's window object.

customClickLoop

customClickLoop replaces the default click loop function so as to provide for scroll bar adjustment in concert with automatic scrolling. Following a mouse-down within the view rectangle, customClickLoop is called repeatedly by TEClick as long as the mouse button remains down.

The first three lines retrieve a handle to the TextEdit structure associated with the window. The next two lines save the current graphics port and set the window's graphics port as the current port.

The window's current clip region will have been set by TextEdit to be equivalent to the view rectangle. Since the scroll bar has to be redrawn, the clipping region must be temporarily reset to include the scroll bar. Accordingly, GetClip saves the current clipping region and the following two lines set the clipping region to the bounds of the coordinate plane.

GetMouse gets the current position of the cursor. If the cursor is above the top of the port rectangle, the text must be scrolled downward. Accordingly, the variable linesToScroll is set to 1. The subsidiary function doSetScrollBarValue is then called to, amongst other things, reset the scroll bar's value. Note that the value in linesToScroll may be modified by doSetScrollBarValue. If linesToScroll is not set to 0 by doSetScrollBarValue, TEScroll is called to scroll the text by a number of pixels equivalent to the value in the lineHeight field of the TextEdit structure, and in a downwards direction.

If the cursor is below the bottom of the port rectangle, the same process occurs except that the variable linesToScroll is set to -1, thus causing an upwards scroll of the text (assuming that the value in linesToScroll is not changed to 0 by doSetScrollBarValue).

If scrolling has occurred, doDrawDataPanel redraws the data panel. SetClip restores the clipping region to that established by the view rectangle and SetPort restores the saved graphics port. Finally, the last line returns true. (A return of false would cause TextEdit to stop calling customClickLoop, as if the user had released the mouse button.)

doSetScrollBarValue

doSetScrollBarValue is called from customClickLoop. Apart from setting the scroll bar's value so as to cause the scroll box to follow up automatic scrolling, the function checks whether the limits of scrolling have been reached.

The first two lines get the current control value and the current control maximum value. At the next block, the value in the variable linesToScroll will be set to either 0 (if the current control value is 0) or equivalent to the control maximum value (if the current control value is equivalent to the control maximum value). If these modifications do not

occur, the value in linesToScroll will remain as established at the first line in this block-that is, the current control value minus the value in `linesToScroll` as passed to the function.

`SetControlValue` sets the control's value to the value in `linesToScroll`. The last line sets the value in `linesToScroll` to 0 if the limits of scrolling have already been reached, or to the value as it was when the `doSetScrollBarValue` function was entered.

doAdjustMenus

`doAdjustMenus` adjusts the menus. Much depends on whether any windows are currently open.

If at least one window is open, the first three lines in the if block get a handle to the TextEdit structure associated with the front window and the first call to `EnableMenuItem` enables the Close item in the File menu. If there is a current selection range, the Cut, Copy, and Clear items are enabled, otherwise they are disabled. If there is data of flavor type 'TEXT' in the scrap (the call to `GetScrapFlavorFlags`), the Paste item is enabled, otherwise it is disabled. If there is any text in the TextEdit structure, the SaveAs and Select All items are enabled, otherwise they are disabled.

If no windows are open, the Close, SaveAs, Clear, and Select All items are disabled.

doMenuChoice

If the MonoTextEdit Help item in the Mac OS 8/9 Apple menu or Mac OS X Help menu is chosen, the function `doHelp` is called.

doFileMenu

`doFileMenu` handles File menu choices, calling the appropriate functions according to the menu item chosen. In the SaveAs case, a handle to the TextEdit structure associated with the front window is retrieved and passed as a parameter to the function `doSaveAsFile`.

Note that, because TextEdit, rather than file operations, is the real focus of this program, the file-related code has been kept to a minimum, even to the extent of having no Save-related, as opposed to SaveAs-related, code.

doEditMenu

`doEditMenu` handles choices from the Edit menu. Recall that, in the case of monostyled TextEdit structures, `TECut`, `TECopy`, and `TEPaste` do not copy/paste text to/from the

scrap. This program, however, supports copying/pasting to/from the scrap.

Before the usual switch is entered, a handle to the TextEdit structure associated with the front window is retrieved.

The `iCut` case handles the Cut command. Firstly, the call to `ClearCurrentScrap` attempts to clear the scrap. If the call succeeds, `PurgeSpace` establishes the size of the largest block in the heap that would be available if a general purge were to occur. The next line gets the current selection length. If the selection length is greater than the available memory, the user is advised via an error message. Otherwise, `TECut` is called to remove the selected text from the TextEdit structure and copy it to the TextEdit private scrap. The scroll bar is adjusted, and `TEToScrap` is called to copy the private scrap to the scrap. If the `TEToScrap` call is not successful, `ClearCurrentScrap` cleans up as best it can by emptying the scrap.

The `iCopy` case handles the Copy command. If the call to `ClearCurrentScrap` to empty the scrap is successful, `TECopy` is called to copy the selected text from the TextEdit structure to the TextEdit private scrap. `TEToScrap` then copies the private scrap to the scrap. If the `TEToScrap` call is not successful, `ClearCurrentScrap` cleans up as best it can by emptying the scrap.

The `iPaste` case handles the Paste command, which must not proceed if the paste would cause the TextEdit limit of 32,767 bytes to be exceeded. The third line establishes a value equal to the number of bytes in the TextEdit structure plus the number of bytes of the 'TEXT' flavor type in the scrap. If this value exceeds the TextEdit limit, the user is advised via an error message. Otherwise, `TEFromScrap` copies the scrap to TextEdit's private scrap, `TEPaste` inserts the private scrap into the TextEdit structure, and the following line adjusts the scroll bar.

The `iClear` case handles the Clear command. `TEDelete` deletes the current selection range from the TextEdit structure and the following line adjusts the scroll bar.

The `iSelectAll` case handle the Select All command. `TESetSelect` sets the selection range according to the first two parameters (`selStart` and `selEnd`).

doGetSelectLength

`doGetSelectLength` returns a value equal to the length of the current selection.

doAdjustScrollbar

`doAdjustScrollbar` adjusts the vertical scroll bar.

The first two lines retrieve handles to the document structure and TextEdit structure

associated with the window in question.

At the next block, the value in the `nLines` field of the TextEdit structure is assigned to the `numberOfLines` variable. The next action is somewhat of a refinement and is therefore not essential. If the last character in the TextEdit structure is the return character, `numberOfLines` is incremented by one. This will ensure that, when the document is scrolled to its end, a single blank line will appear below the last line of text.

At the next block, the variable `controlMax` is assigned a value equal to the number of lines in the TextEdit structure less the number of lines that will fit in the view rectangle. If this value is less than 0 (indicating that the number of lines in the TextEdit structure is less than the number of lines that will fit in the view rectangle), `controlMax` is set to 0. `SetControlMaximum` then sets the control maximum value. If `controlMax` is 0, the scroll bar is automatically unhighlighted by the `SetControlMaximum` call.

The first line of the next block assigns to the variable controlValue a value equal to the number of text lines that the top of the destination rectangle is currently "above" the top of the view rectangle. If the calculation returns a value less than 0 (that is, the document has been scrolled fully down), `controlValue` is set to 0. If the calculation returns a value greater than the current control maximum value (that is, the document has been scrolled fully up), `controlValue` is set to equal that value. `SetControlValue` sets the control value to the value in `controlValue`. For example, if the top of the view rectangle is 2, the top of the destination rectangle is -34 and the `lineHeight` field of the TextEdit structure contains the value 13, the control value will be set to 3.

`SetControlViewSize` is called to advise the Control Manager of the height of the view rectangle. This will cause the scroll box/scroller to be a proportional scroll box/ scroller. (On Mac OS 8/9, this assumes that the user has selected Smart Scrolling on in the Appearance control panel.)

With the control maximum value and the control value set, `TEScroll` is called to make sure the text is scrolled to the position indicated by the scroll box/scroller. Extending the example in the previous paragraph, the second parameter in the `TEScroll` call is 2 - (34 - (3 * 13)), that is, 0. In that case, no corrective scrolling actually occurs.

doAdjustCursor

`doAdjustCursor` is called when the `kEventMouseMoved`, and `kEventWindowActivated` event types are received. It adjusts the cursor to the I-Beam shape when the cursor is over the content region less the scroll bar area, and to the arrow shape when the cursor is outside that region. It is similar to the cursor adjustment function in the demonstration program `GworldPicCursIcn` (Chapter 13).

doCloseWindow

`doCloseWindow` is called when the `kEventWindowClose` event type is received and when the user chooses Close from the File menu. It disposes of the specified window. The associated scroll bar, the associated TextEdit structure and the associated document structure are disposed of before the call to `DisposeWindow`.

doSaveAsFile, doOpenCommand, doOpenFile

The functions `doSaveAsFile`, `doOpenCommand`, and `doOpenFile` are document saving and opening functions, enabling the user to open and save 'TEXT' documents. Since the real focus of this program is TextEdit, not file operations, the code is "bare bones" and as brief as possible, Navigation Services 2.0 functions being used rather than the Navigation Services 3.0 functions used in the demonstration program Files (see Chapter 18).

> For a complete example of opening and saving monostyled 'TEXT' documents, see the demonstration program at Files (see Chapter 18).

doDrawDataPanel

`doDrawDataPanel` draws the data panel at the bottom of each window. Displayed in this panel are the values in the `teLength`, `nLines`, `lineHeight` and `destRect.top` fields of the TextEdit structure and the `contrlValue` and `contrlMax` fields of the scroll bar's control structure.

HelpDialog.c

defines

Constants are established for the index of error strings within a `'STR#'` resource and 'TEXT', `'styl'`, and `'PICT'` resource IDs. `kTextInset` which will be used to inset the view and destination rectangles a few pixels inside a user pane's rectangle.

typedef

The first two data types are for a picture information structure and a document structure. (Note that one field in the document structure is a pointer to a picture information structure.) The third data type will be used for saving and restoring the background color and pattern.

doHelp

`doHelp` is called when the user chooses the MonoTextEdit Help item in the Help menu.

The dialog will utilize a window event handler, a user pane drawing (callback) function, and a control action (callback) function. The first block creates the associated universal procedure pointers.

The call to `CreateNewWindow` creates a new window of the movable modal class. `NewHandle` creates a block for a document structure and the handle is stored in the window object.

The next block creates the dialog's controls and assigns an ID to each. In the case of the user pane control, `SetControlData` is called to set a user pane drawing function. In the case of the scroll bar, the control reference is assigned to the appropriate field of the dialog's document structure.

At the next block, the destination and view rectangles are both made equal to the user pane's rectangle, but inset four pixels from the left and right and two pixels from the top and bottom. The call to `TEStyleNew` creates a multistyled TextEdit structure based on those two rectangles.

A pointer to a picture information structure will eventually be assigned to a field in the document structure. For the moment, that field is set to `NULL`.

At the next block, two global variables are assigned the resource IDs relating to the first Help topic's 'TEXT'/`'styl'` resource and associated `'PICT'` resources.

The next block calls the function `doGetText` that loads the specified 'TEXT'/`'styl'` resources and inserts the text and style information into the TextEdit structure.

The next block calls the function `doGetPictureInfo` that searches for option-space characters in the 'TEXT' resource and, if option-space characters are found, loads a like number of `'PICT'` resources beginning with the specified ID.

`NewRgn` creates an empty region, which will be used to save the dialog's graphic's port's clipping region.

To complete the initial setting up, `ShowWindow` is called to make the dialog visible, following which `RunAppModalLoopForWindow` is called to run the nodal loop.

helpWindowEventHandler

`helpWindowEventHandler` is the event handler for the dialog. It responds to mouse clicks in the dialog's three controls.

If the push button was clicked, `QuitAppModalLoopForWindow` is called to terminate the modal loop and restore menu activation/deactivation status to that which obtained prior to the call to `RunAppModalLoopForWindow`, and the dialog is closed down.

Note that if the click was in the scroll bar, `TrackControl` is called with a universal procedure pointer to an application-defined action (callback) function is passed in the `actionProc` parameter.

If the click was in the pop-up menu button, the menu item chosen is determined, and the switch assigns the appropriate 'TEXT'/`'styl'` and `'PICT'` resource IDs to the global variables which keep track of which of those resources are to be loaded and displayed.

The next three lines get the view rectangle from the TextEdit structure, allowing the next blocks to perform the same "get text" and "get picture information" actions as were preformed at startup, but this time with the 'TEXT'/`'styl'` and `'PICT'` resources as determined within the preceding switch.

The call to `doDrawPictures` draws any pictures that might initially be located in the view rectangle.

userPaneDrawFunction

`userPaneDrawFunction` is the user pane drawing function set within doHelp.

The first line gets a reference to the user pane control's owning window. The next three lines get the user pane's rectangle, insets that rectangle by one pixel all round, and then further expands it to the right edge of the scroll bar. At the next block, a list box frame is drawn in the appropriate state, depending on whether the the movable modal dialog is currently the active window.

The next block erases the previously defined rectangle with the white color using the white pattern.

The next three lines retrieve the view rectangle from the TextEdit structure. The call to `TEUpdate` draws the text in the TextEdit structure in the view rectangle. The call to `doDrawPictures` draws any pictures that might currently be located in the view rectangle.

doGetText

`doGetText` is called when the dialog is first opened and when the user chooses a new item from the pop-up menu. It loads the 'TEXT'/`'styl'` resources associated with the current menu item and inserts the text and style information into the TextEdit structure.

The first two lines get a handle to the TextEdit structure. The next two lines set the selection range to the maximum value and then delete that selection. The destination rectangle is then made equal to the view rectangle and the scroll bar's value is set to 0.

`GetResource` is called twice to load the specified 'TEXT'/`'styl'` resources, following which `TEStyleInsert` is called to insert the text and style information into the TextEdit structure. Two calls to `ReleaseResource` then release the 'TEXT'/`'styl'` resources.

The next block gets the total height of the text in pixels.

At the next block, if the height of the text is greater than the height of the view rectangle, the local variable `heightToScroll` is made equal to the total height of the text minus the height of the view rectangle. This value is then used to set the scroll bar's maximum value. The scroll bar is then made active.

`SetControlViewSize` is called to advise the Control Manager of the height of the view rectangle. This will cause the scroll box to be a proportional scroll box.

If the height of the text is less than the height of the view rectangle, the scroll bar is made inactive.

`true` is returned if the `GetResource` calls did not return with `false`.

doGetPictureInfo

`doGetPictureInfo` is called after `getText` when the dialog is opened and when the user chooses a new item from the pop-up menu. It searches for option-space characters in the 'TEXT' resource and, if option-space characters are found, loads a like number of `'PICT'` resources beginning with the specified ID.

The first line gets a handle to the dialog's document structure.

If the `picInfoRecPtr` field of the document structure does not contain `NULL`, the currently loaded `'PICT'` resources are released, the picture information structures are disposed of, and the `picInfoRecPtr` field of the document structure is set to `NULL`.

The next line sets to 0 the field of the document structure which keeps track of the number of pictures associated with the current 'TEXT' resource.

The next two lines get a handle to the TextEdit structure, then a handle to the block containing the actual text. This latter is then used to assign the size of that block to a local variable. After two local variables are initialized, the block containing the text is locked.

The next block counts the number of option-space characters in the text block. At the following block, if there are no option-space characters in the block, the block is unlocked and the function returns.

A call to `NewPtr` then allocates a nonrelocatable block large enough to accommodate a number of picture information structures equal to the number of option-space characters found. The pointer to the block is then assigned to the appropriate field of the dialog's document structure.

The next line resets the offset value to 0.

The for loop repeats for each of the option-space characters found. `GetPicture` loads the specified `'PICT'` resource (the resource ID being incremented from the base ID at each pass through the loop) and assigns the handle to the appropriate field of the relevant picture information structure. Munger finds the offset to the next option-space character and `TEGetPoint` gets the point, based on the destination rectangle, of the bottom left of the character at that offset. `TEGetStyle` is called to obtain the line height of the character at the offset and this value is subtracted from the value in the point's v field. The offset is incremented and the rectangle in the picture structure's `picFrame` field is assigned to the bounds field of the picture information structure. The next block then offsets this rectangle so that it is centered laterally in the destination rectangle with its top offset from the top of the destination rectangle by the amount established at the `line picturePoint.v -= lineHeight;`.

The third last line assigns the number of pictures loaded to the appropriate field of the dialog's document structure. The block containing the text is then unlocked. The function returns true if false has not previously been returned within the for loop.

actionFunction

`actionFunction` is the action function called from within the event filter (callback) function `eventFilter`. It is repeatedly called by `TrackControl` while the mouse button remains down within the scroll bar. Its ultimate purpose is to determine the new scrollbar value when the mouse-down is within the scroll arrows or gray areas/track of the scroll bar, and then call a separate function to effect the actual scrolling of the text and pictures based on the new scrollbar value. (The scroll bar is the live scrolling variant, so the CEDF automatically updates the control's value while the mouse remains down in the scroll box/scroller.)

If the cursor is still not within the control, execution falls through to the bottom of the function and the action function exits.

The first block gets a pointer to the owner of the scrollbar, retrieves a handle to the dialog's document structure, gets a handle to the TextEdit structure, gets the view rectangle, and assigns a value to the h field of a point variable equal to the left of the view rectangle plus

4 pixels.

The switch executes only if the mouse-down is not in the scroll box.

In the case of the Up scroll arrow, the variable delta is assigned a value which will ensure that, after the scroll, the top of the incoming line of text will be positioned cleanly at top of the view rectangle.

In the case of the Down scroll arrow, the variable delta is assigned a value which will ensure that, after the scroll, the bottom of the incoming line of text will be positioned cleanly at bottom of the view rectangle.

In the case of the Up gray area/track, the variable delta is assigned a value which will ensure that, after the scroll, the top of the top line of text will be positioned cleanly at the top of the view rectangle and the line of text which was previously at the top will still be visible at the bottom of the view rectangle.

In the case of the Down gray area/track, the variable delta is assigned a value which will ensure that, after the scroll, the bottom of the bottom line of text will be positioned cleanly at the bottom of the view rectangle and the line of text which was previously at the bottom will still be visible at the top of the view rectangle.

The first line after the switch gets the pre-scroll scroll bar value. If the text is not fully scrolled up and a scroll up is called for, or if the text is not fully scrolled down and a scroll down is called for, the current clipping region is saved, the clipping region is set to the dialog's port rectangle, the scroll bar value is set to the required new value, and the saved clipping region is restored. (TextEdit may have set the clipping region to the view rectangle, so it must be changed to include the scroll bar area, otherwise the scroll bar will not be drawn.)

With the scroll bar's new value set and the scroll box redrawn in its new position, the function for scrolling the text and pictures is called. Note that this last line will also be called if the mouse-down was within the scroll box.

doScrollTextAndPicts

`doScrollTextAndPicts` is called from `actionFunction`. It scrolls the text within the view rectangle and calls another function to draw any picture whose rectangle intersects the "vacated" area of the view rectangle.

The first line sets the background color to white and the background pattern to white.

The next two lines get a handle to the TextEdit structure. The next line determines the difference between the top of the destination rectangle and the top of the view rectangle

and the next subtracts from this value the scroll bar's new value. If the result is zero, the text must be fully scrolled in one direction or the other, so the function simply returns.

If the text is not already fully scrolled one way or the other, `TEScroll` scrolls the text in the view rectangle by the number of pixels determined at the fifth line.

If there are no pictures associated with the 'TEXT' resource in use, the function returns immediately after the text is scrolled.

The next consideration is the pictures and whether any of their rectangles, as stored in the picture information structure, intersect the area of the view rectangle "vacated" by the scroll. At the if/else block, a rectangle is made equal to the "vacated" area of the view rectangle, the if block catering for the scrolling up case and the else block catering for the scrolling down case. This rectangle is passed as a parameter in the call to `drawPictures`.

doDrawPictures

`doDrawPictures` determines whether any pictures intersect the rectangle passed to it as a formal parameter and draws any pictures that do.

The first two lines get handles to the dialog's document structure and the TextEdit structure.

The next line determines the difference between the top of the destination rectangle and the top of the view rectangle. This will be used later to offset the picture's rectangle from destination rectangle coordinates to view rectangle coordinates. The next line determines the number of pictures associated with the current 'TEXT' resource, a value which will be used to control the number of passes through the following for loop.

Within the loop, the picture's rectangle is retrieved from the picture information structure and offset to the coordinates of the view rectangle. `SectRect` determines whether this rectangle intersects the rectangle passed to `drawPictures` from `scrollTextAndPicts`. If it does not, the loop returns for the next iteration. If it does, the picture's handle is retrieved, `LoadResource` checks whether the `'PICT'` resource is in memory and, if necessary, loads it, `HLock` locks the handle, `DrawPicture` draws the picture, and `HUnlock` unlocks the handle. Before `DrawPicture` is called, the clipping region is temporarily adjusted to equate to the rectangle passed to `drawPictures` from `scrollTextAndPicts` so as to limit drawing to that rectangle.

doCloseHelp

`doCloseHelp` closes down the Help dialog.

The first two lines retrieve a handle to the dialog's document structure. The next two lines dispose of the region used to save the clipping region. `TEDispose` disposes of the

TextEdit structure. The next block disposes of any `'PICT'` resources currently in memory, together with the picture information structure. Finally, the window's document structure is disposed of, the window itself is disposed of, and the graphics port saved in `doHelp` is set as the current port.

doSetBackgroundWhite

`doSetBackgroundWhite` sets the background color to white and the background pattern to the pattern white.

Demonstration Program DateTimeNumbers Listing

```
//
**********************************************************************************************
// DateTimeNumbers.c                                                        CARBON EVENT
MODEL
//
**********************************************************************************************
//
// This program, which opens a single modeless dialog, demonstrates the formatting and
display
// of dates, times and numbers.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  An 'MBAR' resource, and 'MENU' resources for Apple/Application, File, and Edit menus
//    (preload,  non-purgeable).
//
// •  A 'DLOG' resource and associated 'dlgx', 'DITL', 'dfnt', and 'CNTL' resources
//    (purgeable).
//
// •  'hdlg' and 'STR#' resources (purgeable) for balloon help and help tags.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
//
**********************************************************************************************

// ..........................................................................
includes

#include <Carbon.h>
#include <string.h>

// ..........................................................................
defines

#define rMenubar                128
#define mAppleApplication       128
#define  iAbout                 1
#define mFile                   129
#define  iQuit                  12
#define mEdit                   130
#define  iCut                   3
#define  iCopy                  4
#define  iPaste                 5
#define  iClear                 6
#define rDialog                 128
#define  iStaticTextTodaysDate  2
#define  iStaticTextCurrentTime 4
#define  iEditTextTitle         10
#define  iEditTextQuantity      11
#define  iEditTextValue         12
#define  iEditTextDate          13
#define  iButtonEnter           18
#define  iButtonClear           19
#define  iStaticTextTitle       26
#define  iStaticTextQuantity    27
#define  iStaticTextUnitValue   28
```

```
#define  iStaticTextTotalValue  29
#define  iStaticTextDate        30
#define kReturn                 0x0D
#define kEnter                  0x03
#define kLeftArrow              0x1C
#define kRightArrow             0x1D
#define kUpArrow                0x1E
#define kDownArrow              0x1F
#define kBackspace              0x08
#define kDelete                 0x7F
#define topLeft(r)              (((Point *) &(r))[0])
#define botRight(r)             (((Point *) &(r))[1])

// ..................................................................... global
variables

Boolean         gRunningOnX = false;
DialogRef       gDialogRef;
DateCacheRecord gDateCacheRec;
Boolean         gInBackground;

// ..................................................................... function
prototypes

void                    main                (void);
void                    doPreliminaries     (void);
OSStatus                appEventHandler     (EventHandlerCallRef,EventRef,void *);
OSStatus                windowEventHandler  (EventHandlerCallRef,EventRef,void *);
void                    doIdle              (void);
void                    doMenuChoice        (MenuID,MenuItemIndex);
void                    doCopyPString       (Str255,Str255);
void                    doTodaysDate        (void);
void                    doAcceptNewRecord   (void);
void                    doUnitAndTotalValue (Str255,Str255);
void                    doDate              (Str255);
void                    doAdjustCursor      (WindowRef);
void                    doClearAllFields    (void);
ControlKeyFilterResult  numericFilter       (ControlRef,SInt16 *,SInt16 *,EventModifiers *);
void                    helpTags            (void);
// ****************************************************************************
main

void  main(void)
{
  MenuBarHandle       menubarHdl;
  SInt32              response;
  MenuRef             menuRef;
  ControlKeyFilterUPP numericFilterUPP;
  ControlRef          controlRef;
  EventTypeSpec   applicationEvents[] = {  {  kEventClassApplication,
kEventAppActivated    } ,
                                         {  kEventClassCommand,
kEventProcessCommand  } ,
                                         {  kEventClassMouse,
kEventMouseMoved      }  } ;
  EventTypeSpec   windowEvents[]      = {  {  kEventClassWindow,
kEventWindowDrawContent  } ,
                                         {  kEventClassWindow,
kEventWindowActivated    } ,
                                         {  kEventClassWindow,
kEventWindowClose        } ,
                                         {  kEventClassMouse,
kEventMouseDown          } ,
                                         {  kEventClassKeyboard,
kEventRawKeyDown         }  } ;
```

```
  // ..................................................................... do
preliminaries

  doPreliminaries();

  // ............................................................. set up menu bar and
menus

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    ExitToShell();
  SetMenuBar(menubarHdl);
  DrawMenuBar();

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
      DisableMenuItem(menuRef,0);
    }

    gRunningOnX = true;
  }
  else
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
      SetMenuItemCommandID(menuRef,iQuit,kHICommandQuit);
  }

   // ..................................... install application event handler and a
timer

  InstallApplicationEventHandler(NewEventHandlerUPP((EventHandlerProcPtr) appEventHandler),
                                 GetEventTypeCount(applicationEvents),applicationEvents,
                                 0,NULL);
  InstallEventLoopTimer(GetCurrentEventLoop(),0,TicksToEventTime(GetCaretTime()),
                        NewEventLoopTimerUPP((EventLoopTimerProcPtr) doIdle),NULL,NULL);

  // ........................... open modeless dialog, change attributes, and install
handler

  if(!(gDialogRef = GetNewDialog(rDialog,NULL,(WindowRef) -1)))
    ExitToShell();

  ChangeWindowAttributes(GetDialogWindow(gDialogRef),kWindowStandardHandlerAttribute |
                                                     kWindowCloseBoxAttribute,
                                                     kWindowCollapseBoxAttribute);

  InstallWindowEventHandler(GetDialogWindow(gDialogRef),
                            NewEventHandlerUPP((EventHandlerProcPtr) windowEventHandler),
                            GetEventTypeCount(windowEvents),windowEvents,0,NULL);

  // .... create universal procedure pointers for key filter, attach to two edit text
controls

  numericFilterUPP = NewControlKeyFilterUPP((ControlKeyFilterProcPtr) numericFilter);

  GetDialogItemAsControl(gDialogRef,iEditTextQuantity,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlEditTextKeyFilterTag,
                 sizeof(numericFilterUPP),&numericFilterUPP);
```

```
  GetDialogItemAsControl(gDialogRef,iEditTextValue,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlEditTextKeyFilterTag,
                 sizeof(numericFilterUPP),&numericFilterUPP);

  // ......................... set help tags, get and display today's date and show window

  if(gRunningOnX)
    helpTags();

  doTodaysDate();

  ShowWindow(GetDialogWindow(gDialogRef));

  // ............................... display today's date and initialise date cache
structure

  InitDateCache(&gDateCacheRec);

  // ................................................................ run application event
loop

  RunApplicationEventLoop();
}

// ************************************************************************
doPreliminaries

void  doPreliminaries(void)
{
  MoreMasterPointers(64);
  InitCursor();
}
// ************************************************************************
appEventHandler

OSStatus  appEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                          void * userData)
{
  OSStatus      result = eventNotHandledErr;
  UInt32        eventClass;
  UInt32        eventKind;
  HICommand     hiCommand;
  MenuID        menuID;
  MenuItemIndex menuItem;

  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);

  switch(eventClass)
  {
    case kEventClassApplication:
      if(eventKind == kEventAppActivated)
        SetThemeCursor(kThemeArrowCursor);
      break;

    case kEventClassCommand:
      if(eventKind == kEventProcessCommand)
      {
        GetEventParameter(eventRef,kEventParamDirectObject,typeHICommand,NULL,
                          sizeof(HICommand),NULL,&hiCommand);
        menuID = GetMenuID(hiCommand.menu.menuRef);
        menuItem = hiCommand.menu.menuItemIndex;
        if((hiCommand.commandID != kHICommandQuit) &&
           (menuID >= mAppleApplication && menuID <= mEdit))
        {
          doMenuChoice(menuID,menuItem);
```

```
          result = noErr;
        }
      }
      break;

    case kEventClassMouse:
      if(eventKind == kEventMouseMoved)
      {
        doAdjustCursor(GetDialogWindow(gDialogRef));
        result = noErr;
      }
      break;
  }

  return result;
}
// ************************************************************************
windowEventHandler

OSStatus  windowEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                             void* userData)
{
  OSStatus    result = eventNotHandledErr;
  UInt32      eventClass;
  UInt32      eventKind;
  EventRecord eventRecord;
  SInt16      itemHit;
  SInt8       charCode;
  ControlRef  controlRef;
  UInt32      finalTicks;

  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);

  switch(eventClass)
  {
    case kEventClassWindow:                                          // event class
window
      ConvertEventRefToEventRecord(eventRef,&eventRecord);
      switch(eventKind)
      {
        case kEventWindowActivated:
          DialogSelect(&eventRecord,&gDialogRef,&itemHit);
          result = noErr;
          break;

        case kEventWindowClose:
          QuitApplicationEventLoop();
          result = noErr;
          break;
      }

    case kEventClassMouse:                                           // event class
mouse
      ConvertEventRefToEventRecord(eventRef,&eventRecord);
      switch(eventKind)
      {
        case kEventMouseDown:
          if(IsDialogEvent(&eventRecord))
          {
            if(DialogSelect(&eventRecord,&gDialogRef,&itemHit))
            {
              if(itemHit == iButtonEnter)
              {
                doAcceptNewRecord();
                doClearAllFields();
```

```
            }
            else if(itemHit == iButtonClear)
              doClearAllFields();
          }
        }
        doAdjustCursor(GetDialogWindow(gDialogRef));
        break;
      }
      break;
    case kEventClassKeyboard:                                    // event class
keyboard
      switch(eventKind)
      {
        case kEventRawKeyDown:
          ConvertEventRefToEventRecord(eventRef,&eventRecord);
          GetEventParameter(eventRef,kEventParamKeyMacCharCodes,typeChar,NULL,
                            sizeof(charCode),NULL,&charCode);
          if((charCode == kReturn) || (charCode == kEnter))
          {
            GetDialogItemAsControl(gDialogRef,iButtonEnter,&controlRef);
            HiliteControl(controlRef,kControlButtonPart);
            Delay(8,&finalTicks);
            HiliteControl(controlRef,kControlEntireControl);
            doAcceptNewRecord();
            doClearAllFields();
            return noErr;
          }
          break;
      }
      break;
  }

  return result;
}

// ********************************************************************************
doIdle

void  doIdle(void)
{
  UInt32        rawSeconds;
  static UInt32 oldRawSeconds;
  Str255        timeString;
  ControlRef    controlRef;

  if(gRunningOnX)
    IdleControls(GetDialogWindow(gDialogRef));

  GetDateTime(&rawSeconds);

  if(rawSeconds > oldRawSeconds)
  {
    TimeString(rawSeconds,true,timeString,NULL);

    GetDialogItemAsControl(gDialogRef,iStaticTextCurrentTime,&controlRef);
    SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,timeString[0],
                   &timeString[1]);
    Draw1Control(controlRef);

    oldRawSeconds = rawSeconds;
  }
}
// ********************************************************************************
doMenuChoice

void  doMenuChoice(MenuID menuID,MenuItemIndex menuItem)
```

```
{
  if(menuID == 0)
    return;

  switch(menuID)
  {
    case mAppleApplication:
      if(menuItem == iAbout)
        SysBeep(10);
      break;

    case mEdit:
      switch(menuItem)
      {
        case iCut:
          DialogCut(gDialogRef);
          break;

        case iCopy:
          DialogCopy(gDialogRef);
          break;

        case iPaste:
          DialogPaste(gDialogRef);
          break;

        case iClear:
          DialogDelete(gDialogRef);
          break;
      }
      break;
  }
}

// *****************************************************************************
doCopyPString

void  doCopyPString(Str255 sourceString,Str255 destinationString)
{
  SInt16 stringLength;

  stringLength = sourceString[0];
  BlockMove(sourceString + 1,destinationString + 1,stringLength);
  destinationString[0] = stringLength;
}

// *****************************************************************************
doTodaysDate

void  doTodaysDate(void)
{
  UInt32     rawSeconds;
  Str255     dateString;
  ControlRef controlRef;
  GetDateTime(&rawSeconds);
  DateString(rawSeconds,longDate,dateString,NULL);

  GetDialogItemAsControl(gDialogRef,iStaticTextTodaysDate,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,dateString[0],
                 &dateString[1]);
}

// *****************************************************************************
doAcceptNewRecord

void  doAcceptNewRecord(void)
```

```
{
  SInt16     theType;
  Handle     theHandle;
  Rect       theRect;
  Str255     titleString, quantityString, valueString, dateString;
  ControlRef controlRef;

  GetDialogItem(gDialogRef,iEditTextTitle,&theType,&theHandle,&theRect);
  GetDialogItemText(theHandle,titleString);

  GetDialogItem(gDialogRef,iEditTextQuantity,&theType,&theHandle,&theRect);
  GetDialogItemText(theHandle,quantityString);

  GetDialogItem(gDialogRef,iEditTextValue,&theType,&theHandle,&theRect);
  GetDialogItemText(theHandle,valueString);

  GetDialogItem(gDialogRef,iEditTextDate,&theType,&theHandle,&theRect);
  GetDialogItemText(theHandle,dateString);

  if(titleString[0] == 0 || quantityString[0] == 0 || valueString[0] == 0 ||
     dateString[0] == 0)
  {
    SysBeep(10);
    return;
  }

  GetDialogItemAsControl(gDialogRef,iStaticTextTitle,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,titleString[0],
                 &titleString[1]);
  Draw1Control(controlRef);

  GetDialogItemAsControl(gDialogRef,iStaticTextQuantity,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,quantityString
[0],
                 &quantityString[1]);
  Draw1Control(controlRef);

  doUnitAndTotalValue(valueString,quantityString);

  doDate(dateString);
}
// *********************************************************************
doUnitAndTotalValue

void  doUnitAndTotalValue(Str255 valueString, Str255 quantityString)
{
  Handle           itl4ResourceHdl;
  SInt32           numpartsOffset;
  SInt32           numpartsLength;
  NumberParts      *numpartsTablePtr;
  Str255           formatString = "\p'$'###,###,###.00;'Valueless';'Valueless'";
  NumFormatString  formatStringRec;
  Str255           formattedNumString;
  extended80       value80Bit;
  SInt32           quantity;
  double           valueDouble;
  FormatResultType result;
  ControlRef       controlRef;

  GetIntlResourceTable(smSystemScript,iuNumberPartsTable,&itl4ResourceHdl,&numpartsOffset,
                       &numpartsLength);
  numpartsTablePtr = (NumberPartsPtr) ((SInt32) *itl4ResourceHdl + numpartsOffset);

  StringToFormatRec(formatString,numpartsTablePtr,&formatStringRec);

  StringToExtended(valueString,&formatStringRec,numpartsTablePtr,&value80Bit);
```

```
  ExtendedToString(&value80Bit,&formatStringRec,numpartsTablePtr,formattedNumString);

  GetDialogItemAsControl(gDialogRef,iStaticTextUnitValue,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,
                formattedNumString[0],&formattedNumString[1]);
  Draw1Control(controlRef);

  StringToNum(quantityString,&quantity);

  valueDouble = x80tod(&value80Bit);
  valueDouble = valueDouble * quantity;
  dtox80(&valueDouble,&value80Bit);

  result = ExtendedToString(&value80Bit,&formatStringRec,numpartsTablePtr,
                           formattedNumString);

  if(result == fFormatOverflow)
    doCopyPString("\p(Too large to display)",formattedNumString);

  GetDialogItemAsControl(gDialogRef,iStaticTextTotalValue,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,
                formattedNumString[0],&formattedNumString[1]);
  Draw1Control(controlRef);
}
// *****************************************************************************
doDate

void  doDate(Str255 dateString)
{
  SInt32       lengthUsed;
  LongDateRec  longDateTimeRec;
  LongDateTime longDateTimeValue;
  ControlRef   controlRef;

  longDateTimeRec.ld.hour = 0;
  longDateTimeRec.ld.minute = 0;
  longDateTimeRec.ld.second = 0;

  StringToDate((Ptr) dateString + 1,dateString[0],&gDateCacheRec,&lengthUsed,
              &longDateTimeRec);

  LongDateToSeconds(&longDateTimeRec,&longDateTimeValue);
  LongDateString(&longDateTimeValue,longDate,dateString,NULL);

  GetDialogItemAsControl(gDialogRef,iStaticTextDate,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,dateString[0],
                &dateString[1]);
  Draw1Control(controlRef);
}

// *****************************************************************************
doAdjustCursor

void  doAdjustCursor(WindowRef windowRef)
{
  GrafPtr    oldPort;
  RgnHandle  arrowRegion,iBeamRegion;
  ControlRef controlRef;
  Rect       iBeamRect;
  Point      mouseLocation;

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

  arrowRegion = NewRgn();
  iBeamRegion = NewRgn();
```

```
  SetRectRgn(arrowRegion,-32768,-32768,32767,32767);

  GetKeyboardFocus(windowRef,&controlRef);
  GetControlBounds(controlRef,&iBeamRect);

  LocalToGlobal(&topLeft(iBeamRect));
  LocalToGlobal(&botRight(iBeamRect));

  RectRgn(iBeamRegion,&iBeamRect);
  DiffRgn(arrowRegion,iBeamRegion,arrowRegion);

  GetMouse(&mouseLocation);
  LocalToGlobal(&mouseLocation);
  if(PtInRgn(mouseLocation,iBeamRegion))
    SetThemeCursor(kThemeIBeamCursor);
  else
    SetThemeCursor(kThemeArrowCursor);

  DisposeRgn(arrowRegion);
  DisposeRgn(iBeamRegion);

  SetPort(oldPort);
}

// ************************************************************************
doClearAllFields

void  doClearAllFields(void)
{
  SInt16     a;
  ControlRef controlRef;
  Str255     theString = "\p";

  for(a = iEditTextTitle;a <= iEditTextDate;a++)
  {
    GetDialogItemAsControl(gDialogRef,a,&controlRef);
    SetControlData(controlRef,kControlEntireControl,kControlEditTextTextTag,theString[0],
                &theString[1]);
    Draw1Control(controlRef);

    if(a == iEditTextTitle)
      SetKeyboardFocus(GetDialogWindow(gDialogRef),controlRef,kControlFocusNextPart);
  }
}

// ************************************************************************
numericFilter

ControlKeyFilterResult  numericFilter(ControlRef controlRef,SInt16* keyCode,SInt16
*charCode,
                                      EventModifiers *modifiers)
{
  if(((char) *charCode >= '0') && ((char) *charCode <= '9') || (char) *charCode == '.' ||
     (BitTst(modifiers,15 - cmdKeyBit)))
  {
    return kControlKeyFilterPassKey;
  }

  switch(*charCode)
  {
    case kLeftArrow:
    case kRightArrow:
    case kUpArrow:
    case kDownArrow:
    case kBackspace:
```

```
    case kDelete:
      return kControlKeyFilterPassKey;
      break;
  }

  SysBeep(10);
  return kControlKeyFilterBlockKey;
}
// ********************************************************************************
helpTags

void  helpTags(void)
{
  HMHelpContentRec helpContent;
  SInt16           a;
  static SInt16    itemNumber[7] = {  1,3,21,22,23,24,25 } ;
  ControlRef       controlRef;

  HMSetTagDelay(5);
  HMSetHelpTagsDisplayed(true);
  helpContent.version = kMacHelpVersion;
  helpContent.tagSide = kHMOutsideTopCenterAligned;

  helpContent.content[kHMMinimumContentIndex].contentType = kHMStringResContent;
  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmResID = 128;

  for(a = 1;a <= 7; a++)
  {
    helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = a;
    GetDialogItemAsControl(gDialogRef,itemNumber[a - 1],&controlRef);
    HMSetControlHelpContent(controlRef,&helpContent);
  }
}

//
*********************************************************************************************
```

Demonstration Program DateTimeNumbers Comments

When this program is run, the user should enter data in the four edit text controls, using the tab key or mouse clicks to select the required control and pressing the Return key or clicking the Enter Record button when data has been entered in all controls. Note that numeric filters are used in the Quantity and Value edit text controls.

In order to observe number formatting effects, the user should occasionally enter very large numbers and negative numbers in the Value field. In order to observe the effects of date string parsing and formatting, the user should enter dates in a variety of formats, for example: "2 Mar 95", "2/3/95", "March 2 1995", "2 3 95", and so on.

Global Variables

`gDateCacheRec` is used within the function doDate.

main

The call to `InstallEventLoopTimer` installs a timer which will fire repeatedly at the interval returned by the call to `GetCaretTime`. When the timer fires, the function doIdle is called. In addition to calling `IdleControls`, `doIdle` updates the current time displayed in a static text control in the top of the dialog.

`doTodaysDate` is called to get the date and set it in a static text control at the top of the dialog.

In the function `doDate`, the function that creates the long date-time structure takes an initialized date cache structure as a parameter. The call to `InitDateCache` initializes a date cache structure.

windowEventHandler

Note that all events are passed to `DialogSelect`.

When the `kEventMouseDown` event is received, if the Enter Record push button was hit, the function `doAcceptNewRecord` is called, following which `doClearAllFields` is called to clear all of the edit text controls. The same occurs when the `kEventRawKeyDown` event is received if the key pressed was Return or Enter.

doIdle

`doIdle`, which is called when the timer fires, blinks the insertion point caret and sets the current time in the static text control at top-right in the dialog.

If the program is running on Mac OS 8/9, `IdleControls` is called to ensure that the caret blinks regularly in the edit text control with current keyboard focus. (On Mac OS X, these controls have their own in-built timers.)

`GetDateTime` retrieves the "raw" seconds value, as known to the system. (This is the number of seconds since 1 Jan 1904.) If that value is greater than the value retrieved the last time doIdle was called, `TimeString` converts the raw seconds value to a string containing the time formatted according to flags in the numeric format (`'itl0'`) resource. (Because `NULL` is specified in the resource handle parameter, the appropriate `'itl0'` resource for the current script system is used.) This string is then set in the static text control, following which `Draw1Control` is called to redraw the control. The retrieved raw seconds value is assigned to the static variable `oldRawSeconds` for use next time `doIdle` is called.

doTodaysDate

`doTodaysDate` sets the date in the static text control at top-left of the dialog.

`GetDateTime` gets the raw seconds value, as known to the system. `DateString` converts the raw seconds value to a string containing a date formatted in long date format according to flags in the numeric format (`'itl0'`) resource. (Because `NULL` is specified in the resource handle parameter, the appropriate `'itl0'` resource for the current script system is used.) This string is then set in the static text control.

doAcceptNewRecord

`doAcceptNewRecord` is called when the Return or Enter key is pressed, or when the Enter Record button is clicked. Assuming each edit text control contains at least one character of text, it calls other functions to format (where necessary) and display strings in the "Last Record Entered" group box area.

The calls to `GetDialogItem` get the handle in the `hText` field of each edit text control's TextEdit structure, allowing the calls to `GetDialogItemText` to get the text into four local variables of type `Str255`.

If the length of any of these strings is 0, the system alert sound is played and `doAcceptNewRecord` returns.

The text from the Item Title and Quantity edit text controls are set in the relevant static text controls within the Last Record Entered group box, and `DrawIControl` is called to draw

those controls. doUnitAndTotalValue and doDate are then called.

doUnitAndTotalValue

doUnitAndTotalValue is called by doAcceptNewRecord to convert the string from the Value edit text control to a floating point number, convert that number to a formatted number string, set that string in the relevant static text control, convert the string from the Quantity edit text control to an integer, multiply the floating point number by the integer to arrive at the "Total Value" value, convert the result to a formatted number string, and set that string in the relevant static text control.

A pointer to a number parts table is required by the functions that convert between floating point numbers and strings. Accordingly, the first three lines get the required pointer.

StringToFormatRec converts the number format specification string into the internal numeric representation required by the functions that convert between floating point numbers and strings.

StringToExtended converts the received Value string into a floating point number of type extended (80 bits). ExtendedToString converts that number back to a string, formatted according to the internal numeric representation of the number format specification string. That string is then set in the relevant static text control and Draw1Control is called to draw that control.

The intention now is to multiply the quantity by the unit value to arrive at a total value. The string received in the quantityString formal parameter is converted to an integer value of type SInt32 by StringToNum. The extended80 value is converted to a value of type double before the multiplication occurs. The result of the multiplication is assigned to the variable of type double. This is then converted to an extended80.

The extended80 value is then passed in the first parameter of ExtendedToString for conversion to a formatted string. If ExtendedToString does not return fFormatOverflow, the formatted string is set in the relevant static text control and Draw1Control is called to draw that control.

doDate

doDate is called by doAcceptNewRecord to create a long date-time structure from the string in the "Date" edit text control, format the date as a string (long date format), and set that string in the relevant static text control.

A pointer to the string containing the date as entered by the user, and the length of that string, are passed in the call to StringToDate. StringToDate parses the input string and fills in the relevant fields of the long date-time structure.

The function `StringToDate` fills in only the year, month, day, and day of the week fields of a long date-time structure. The function `StringToTime` fills in the hour, minute, and second. If you do not call `StringToTime`, as is the case here, you need to zero the time-related fields of the long date-time structure. If this is not done, the call to `LongDateToSeconds` will return an erroneous value. (`LongDateToSeconds` always assumes that all the fields of the long date-time structure passed to it are valid.)

`LongDateToSeconds` converts the long date-time structure to a long date-time value. The long date-time value is then passed as a parameter to `LongDateString`, which converts the long date-time value to a long format date string formatted according to the specified international resource. (In this case, `NULL` is passed as the international resource parameter, meaning that the appropriate `'itl1'` resource for the current script system is used.)

The formatted date string is then set in the relevant static text control and `Draw1Control` is called to draw that control.

Chapter 22. Lists and Custom List Definition Functions

IN THIS CHAPTER

- Introduction to Lists
- Appearance and Features of Lists
- Selection of Cells Using the Mouse
- Selection of Cells Using the Keyboard
- **Creating, Disposing Of, and Managing Lists**
- The List Box Control
- Custom List Definition Functions
- Main List Manager Constants, Data Types, and Functions
- Relevant Text Utilities Data Type and Functions
- Demonstration Program Lists Listing
- Demonstration Program Lists Comments

Introduction to Lists

If you need the user to be able to select a single item from a small group of items, you typically provide a pop-up menu. Pop-up menus, however, do not allow the user to select multiple items from a group of items, are not especially suitable for the presentation of large numbers of items, and cannot present items in columns as well as rows. Furthermore, the items in a pop-up menu remain displayed only as long as the user holds the mouse button down.

By using **lists** to present a group of items to the user, you can overcome these limitations. Although lists, like pop-up menus, are generally used to solicit the user's choices, they can also be used to simply present information. Perhaps the most familiar example of such a list is that at the bottom of the window opened when you choose About This Computer from the Mac OS 8/9 Apple menu.

In essence, then, the List Manager allows you to create either one-column or multicolumn scrollable lists which may be used to simply present items of information or, as is most often the case, to enable the user to select one or more of a group of items.

By default, the List Manager creates lists that contain only monostyled text. However, with a little additional effort, you can create lists which display items graphically (as does the list on the left side of the window opened when you choose Chooser from the Mac OS 8/9 Apple menu), or which display more than one type of information in each item (as does the list in the Mac OS 8/9 About This Computer window).

List Manager Limitations

The List Manager is not suitable for displaying large amounts of data. The limiting size for list data is 32KB, and performance degrades well before that limit is reached.

Options for Creating and Managing Lists

You can use the List Manager function `LNew` to create a list, in which case you must provide all the functions necessary to add rows and data to the list, handle mouse and keyboard events, and so on. Alternatively, you can use the **list box control** to simplify the matter of list creation and handling.

The first section of this chapter addresses the former method and the subject of lists in general. The second section addresses the list box control, and indicates those areas in the first section which are not relevant when you use list box controls.

Appearance and Features of Lists

Fig 1 shows a dialog with two typical single-column lists. The items in the list on the left are exclusively text items and the items in the list on the right are recorded pictures comprising a graphic and a title string. The list on the left supports the selection of multiple items.

Figure 1. DIALOG BOX WITH TWO LISTS



To create a list with graphical elements, such as the list at the right in Fig 1, you must write a custom **list definition function,** because the default list definition function only supports the display of text.

Cells

A list is a number of items displayed within a rectangle, each item being contained within an invisible rectangular **cell.** Cells may contain different types of data, but all cells within a particular list are of the same size.

Cell Font

By default, lists inherit the font of the graphics port associated with the window or dialog in which they reside. Ordinarily, your text-only lists should use the large or small system font.

> **Note**
>
> In the case of list control boxes, the font may be set using `SetControlFontStyle` or a `'dftb'` resource.

Regardless of the font your application uses, if a string is too long to fit in its cell using the current font, the List Manager uses condensed type in an attempt to make it fit (Mac OS 8/9 only). Failing that, the List Manager truncates the string and appends the ellipsis character.

Cell Highlighting

Your application may or may not allow the user to select one or more cells in a list. If your application allows users to select cells, then, when the user selects a cell, the List Manager automatically highlights that cell.

Scroll Bars

Lists may contain scroll bars, which allow you to include more items in a list than can be contained within the list's display rectangle. If a list includes a scroll bar but the number of cells is such that they are all visible, the List Manager disables the scroll bar.

Selection of Cells Using the Mouse

LClick

Your application must call `LClick` whenever a mouse-down occurs in an active list. `LClick` handles all user interaction within the list until the user releases the mouse button. This includes cell highlighting and, when the user drags the mouse outside the list's display rectangle, automatic list scrolling. `LClick` also examines the state of the Shift and Command keys, which are central to the process of multiple cell selection in lists.

Multiple Cell Selection Using the Default Cell-Selection Algorithm

The List Manager's cell-selection algorithm allows the user to select a contiguous range of cells, or even several discontiguous ranges of cells, by using the Shift and Command keys in conjunction with the mouse. The following describes the default cell-selection behavior.

> **Note**
>
> If the user presses both the Shift and Command keys when clicking a cell, the Shift key is ignored.

Cell Selection with the Shift Key

The user can extend a selection of just one cell to several contiguous cells by pressing the Shift key and clicking another item. By clicking and dragging with the Shift key down, the user can extend or shrink the range of selected cells. If the cursor is dragged outside the list's display rectangle, the list will scroll so as to enable the user to include cells which were not initially visible.

Cell Selection with the Command Key

A range of cells may be added or deleted from the current selection by pressing the Command key and then dragging the cursor over other cells. The List Manager checks the status of the first cell clicked in so as to determine whether to add or remove selections. All cells in the range over which the cursor passes will be deselected if that first cell is initially selected. On the other hand, all cells in the range over which the cursor passes will be selected if that first cell is initially not selected.

When a cell's selection status is changed by Command-dragging, that selection status remains the same for the duration of the drag-that is, it will not change if the user moves the cursor back over the cell. The effect of the Command key thus differs from that of the Shift key in this respect.

Shift-Clicking-Noncontiguous Cells Selected

Noncontiguity is lost if the user Shift-clicks a cell after having previously created noncontiguous selections. The List Manager selects all cells in the range of the selected cell closest to the top of the list and the newly selected cell-unless the newly selected cell precedes the first selected cell, in which case the List Manager selects all cells in the range of the newly selected cell and the selected cell closest to the bottom of the list.

Customizing the Cell-Selection Algorithm

As will be seen, the List Manager's cell-selection algorithm may easily be customized so as to modify its default behavior. The most common modification is to defeat multiple cell selection, allowing the user to select only one cell.

Selection of Cells Using the Keyboard

Some users prefer to use the keyboard to select cells in lists. Your application should support the following methods of cell selection via the keyboard:

- Cell Selection Using Arrow Keys. This method involves the use of the Arrow keys to move and extend cell selections.
- Type Selection. This method involves the user simply typing the text associated with an item. It is thus only relevant to text-only lists (or lists whose items can be identified by text strings).

Cell Selection Using Arrow Keys

The List Manager does not provide any functions to support cell selection by Arrow key. Accordingly, your application must supply all of the necessary code. The following describes what that code should do.

Moving the Selection Using Arrow Keys

Shift and Command Keys Not Down

When the user presses an Arrow key, and is not at the same time pressing the Shift or Command key, the selection should be moved by one cell.

If the user presses the Up Arrow, for example, your application should respond by selecting the cell which is above the first selected cell and by deselecting all other selected cells. (Of course, if the first selected cell is the topmost cell in the list, your application should respond by simply deselecting all cells other than the first selected cell.) If necessary, your application should then scroll the list to ensure that the newly selected cell is visible.

Command Key Down

When the user presses an Arrow key while at the same time pressing the Command key, your application should move the first selected cell or the last selected cell (depending on which arrow key is used) as far as it can move in the appropriate direction. For example, in a single-column list, pressing the Up Arrow key should select the first cell in the list, deselect all other cells, and scroll the list, if necessary, to ensure that the newly selected cell is visible.

Extending the Selection Using Arrow Keys

When the user presses an Arrow key while the Shift key is down, the user is attempting to **extend** the selection. There are two alternative algorithms your application can use to respond to Shift-Arrow key combinations: the **extend algorithm** and the **anchor algorithm.** The easiest one to implement is the extend algorithm.

The Extend Algorithm

Using the extend algorithm, your application simply finds the first (or last) selected cell, and then selects

another cell in the direction of the Arrow key. For example, if the user presses Shift-Down Arrow in a single-column list, the application should find the last selected cell and select the cell immediately below it, or, if the user presses Shift-Up Arrow, the application should find the first selected cell and select the cell above it. As always, the list should then be scrolled, if necessary, to make the newly selected cell visible.

Type Selection

The List Manager does not provide any functions to support type selection, although the Text Utilities provide a data type and functions which support this method of keyboard selection. That said, your application must provide most of the code. The following describes what that code should do.

In a text-only list, when the user types the text of an item in a list, the list should scroll to that cell and select it.

However, rather than requiring the user to type the entire text of the item before searching for a match, your application should repeatedly search for a match as each character is entered. Accordingly, every time the user types a character, your application should add it to a string. If this string is currently two characters long, for example, your application should then walk the cells of the list, comparing these two characters with the first two characters of the text in each cell. If a match is found, that cell should be selected and the list scrolled, if necessary, to make the cell visible.

Your application should automatically reset the internal string to a null string when the user has not pressed a key for a given amount of time. To make your application consistent with other applications and the Finder, this time should be twice the number of ticks contained in the low memory global `KeyThresh`. (The value in `KeyThresh` is set by the user at the Delay Until Repeat section of the Keyboard control panel (Mac OS 8/9) and System Preferences/Keyboard (Mac OS X).)

Implementing Type Selection

To implement type selection, your application must store the characters the user has typed and the time when the user last typed a character. The Text Utilities `TypeSelectRecord` structure may be used for this purpose:

```
struct TypeSelectRecord
{
  unsigned long tsrLastKeyTime;  // Time when the last character was typed
  ScriptCode    tsrScript;
  Str63         tsrKeyStrokes;   // Characters typed by the user
};
typedef struct TypeSelectRecord TypeSelectRecord;
```

The Text Utilities function `TypeSelectNewKey` adds a new character to the `tsrKeyStrokes` field.

A global variable of type `SInt16` should be used to store the number of ticks after which type selection must be reset. The low memory accessor function `LMGetKeyThresh` might be used to obtain the value in the low memory global `KeyThresh`.

The Text Utilities function `TypeSelectClear` might be used to reset the `tsrKeyStrokes` and `tsrLastKeyTime` fields of the `TypeSelectRecord` structure to `NULL` and `0` respectively.

`LSearch` should be called to search the list's cells for a match with the specified characters, a universal

procedure pointer to a comparison function being passed in the `searchProc` parameter.

Creating, Disposing of, and Managing Lists

The List Structure

The **list structure,** which the List Manager uses to keep track of information about a list, is central to the creation and management of lists. Although the list structure is not opaque, you are encouraged to access the list structure indirectly using the provided accessor functions because this will give your application greater threading capability on Mac OS X.

Before describing the list structure and its associated accessor functions, however, it is necessary to describe another data type used exclusively by the List Manager-that is, the `Cell` data type.

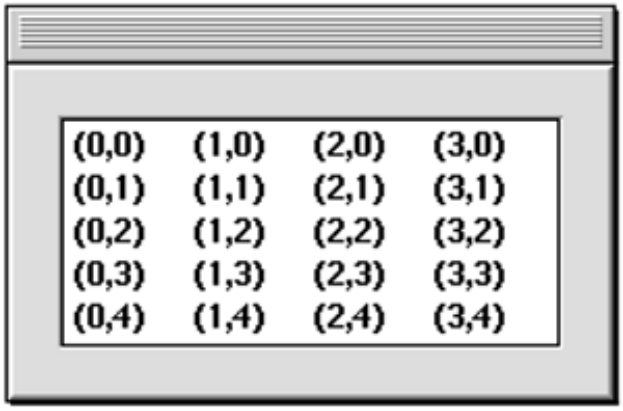
The Cell Data Type

A cell in a list can be described by the `Cell` data type, which has the same structure as the `Point` data type:

```
typedef Point Cell;
```

The `Cell` data type's fields, however, have a different meaning from those of the `Point` data type. In the `Cell` data type, the `h` field specifies the row number and the `v` field specifies the column number. The first cell in a list is defined as cell (0,0). Fig 2 shows a multicolumn list in which each cell's text is set to the coordinates of the cell.

Figure 2. CELL COORDINATES



The ListRec Structure

The list structure is defined by the `ListRec` data type:

```
struct ListRec
{
  Rect              rView;          // List's display rectangle.
  GrafPtr           port;           // List's graphics port.
  Point             indent;         // Indent distance for drawing.
  Point             cellSize;       // Size in pixels of a cell.
  Rect              visible;        // Boundary of visible cells.
  ControlRef        vScroll;        // Vertical scroll bar.
  ControlRef        hScroll;        // Horizontal scroll bar.
  SInt8             selFlags;       // Selection flags.
  Boolean           lActive;        // true if list is active.
  SInt8             lReserved;      // (Reserved.)
  Sint8             listFlags;      // Automatic scrolling flags.
  long              clikTime;       // TickCount at time of last tick.
  Point             clikLoc;        // Position of last click.
```

```
  Point            mouseLoc;      // Current mouse location.
  ListClickLoopUPP lClickLoop;    // Function called by LClick.
  Cell             lastClick;     // Last cell clicked.
  long             refCon;        // For application use.
  Handle           listDefProc;   // List definition function.
  Handle           userHandle;    // For application use.
  ListBounds       dataBounds;    // Boundary of cells allocated.
  DataHandle       cells;         // Cell data.
  short            maxIndex;      // (Used internally.)
  short            cellArray[1];  // Offsets to data.
};

typedef struct ListRec ListRec;
typedef ListRec *ListPtr;
typedef ListPtr *ListHandle;
```

Field Descriptions and Associated Accessor Functions

`rView` The list's visible rectangle (local coordinates). This does not include the area occupied by the list's scroll bars (if any).

Accessor Functions: `GetListViewBounds` and `SetListViewBounds`

`port` The graphics port of the window containing the list. The coordinates of the list's visible rectangle are local to this port.

Accessor Functions: `GetListPort` and `SetListPort`

`indent` The location, relative to the upper left corner of the cell, at which drawing should begin. For example, the default list definition function sets the vertical coordinate of this field to near the bottom of the cell so that characters drawn with QuickDraw's `DrawText` function are centerd vertically in the cell.

Accessor Functions: `GetListCellIndent` and `SetListCellIndent`

`CellSize` The size (in pixels) of each cell in the list. For text-only lists, you usually let the List Manager automatically calculate the cell dimensions. In this case, the List Manager adds the ascent, descent and leading of the port's font to arrive at the height of a cell (which works out as 16 pixels for 12-point Charcoal on Mac OS 8/9, for example). You should make the height of your list equal to a multiple of cell height. For cell width, the List Manager divides the width of the list's display rectangle by the number of columns in the list.

Accessor Functions: `GetListCellSize` and `SetListCellSize`

`visible` Specifies those cells in a list that are visible within the `rView` rectangle. The `left` and `top` fields are set by the List Manager to the coordinates of the first visible cell. The `right` and `bottom` fields are set to one greater than the horizontal and vertical coordinates of the last visible cell. If, for example, the first three columns and six rows are visible (that is, the last visible cell has coordinates (2,5)), the List Manager sets the `visible` field to (0,0,3,6).

The List Manager sets the `right` and `bottom` fields to one greater than the horizontal and vertical coordinates of the last visible cell so as to facilitate the use of QuickDraw's `PtInRect` function to determine whether a cell is currently visible. When `PtInRect` is used for this purpose, a `Cell` variable is passed as the first parameter and the `visible` field is passed as the second parameter. When `PtInRect`'s parameters are expressed as cell coordinates, the cells "hang" down and to the right of the mathematical rectangle. Thus, in the above example, if the cell passed as the first parameter to `PtInRect` specifies row 6 or higher or column 3 or higher, `PtInRect` returns `false`.

The fact that the `visible` field is set in this way also means that the number of visible rows and columns may be determined by simply subtracting the value in the `top` field from the value in the `bottom` field (rows) and the value in the `left` field from the value in the `right` field (columns).

Accessor Functions: `GetListVisibleCells` and `SetListVisibleCells`

`vScroll` Handle to the vertical scroll bar (or `NULL` if there is no vertical scroll bar).

Accessor Functions: GetListVerticalScrollBar and SetListVerticalScrollBar

hScroll Handle to the horizontal scroll bar (or NULL if there is no horizontal scroll bar).

Accessor Functions: GetListHorizontalScrollBar and SetListHorizontalScrollBar

selFlags The algorithm the List Manager uses to select cells in response to a click in the list.

Accessor Functions: GetListSelectionFlags and SetListSelectionFlags

lActive true if a list is active or false if it is inactive. (Do not change this field directly. Use LActivate to activate or deactivate a list.)

Accessor Functions: GetListActive and LActivate

listFlags Flags indicating whether automatic vertical and horizontal scrolling is enabled. If automatic scrolling is enabled, the list scrolls when the user clicks a cell and then drags the cursor out of the rView rectangle. If the list has the associated scroll bar (horizontal or vertical), automatic scrolling is enabled by default. The following constants are used to specify whether horizontal and vertical autoscrolling should be enabled or disabled:

```
lDoVAutoscroll = 2   Allows vertical scrolling.
lDoHAutoscroll = 1   Allows horizontal scrolling.
```

Accessor Functions: GetListFlags and SetListFlags

clikTime The time when the user last clicked the mouse.

Accessor Functions: GetListClickTime and SetListClickTime

clikLoc The local coordinates of the last mouse click.

Accessor Function: GetListClickLocation

mouseLoc urrent location of the cursor in local coordinates.

Accessor Function: GetListMouseLocation

lClickLoop Contains NULL if the default click loop function is to be used. However, your application can assign a universal procedure pointer to a custom click-loop function to this field. Your application will not need a custom click-loop function unless it needs to perform some special processing while the user drags the cursor.

Accessor Functions: GetListClickLoop and SetListClickLoop

lastClick Cell coordinates of the last click. You can access the value in this field using LLastClick.

Accessor Function: SetListLastClick

refCon For your application's use.

Accessor Functions: GetListRefCon and SetListRefCon

listDefProc Handle to the list definition function code.

Accessor Functions: GetListDefinition and SetListDefinition

`userHandle` For your application's use. Typically, applications use this field to store a handle to some additional storage associated with a list.

Accessor Functions: `GetListUserHandle` and `SetListUserHandle`

`dataBounds` The total cell dimensions of the list. It is similar to the `visible` field in that its `right` and `bottom` fields are each set to one greater than the horizontal and vertical coordinates of the last cell, except that, in this case, the "last cell" is the last cell in the list, not the last cell in the visible rectangle. For example, if a list contains 5 columns and 12 rows (that is, the last cell in the list has coordinates (4,10)), the `dataBounds` field is set to (0,0,5,12).

Accessor Function: `GetListDataBounds`

`cells` Handle to a relocatable block where cell data is stored. Because of the way this field is defined, no list can contain more than 32,000 bytes of data.

Accessor Functions: `GetListDataHandle`

`cellArray` Offsets to data in the relocatable block specified by the `cells` field. Do not change the `cells` field, or access the information in the `cellArray` field, directly.

The fields of a list structure that you will be most concerned with are the `rView`, `port`, `cellSize`, `visible`, and `dataBounds` fields.

Creating Lists

Creating Lists That Do Not Use a Custom List Definition Function

If you are creating a list that does not use a custom list definition function, you should use the function `LNew` to create the list:

```
ListHandle  LNew(const Rect *rView,const ListBounds *dataBounds,Point cSize,
                 short theProc,WindowRef theWindow,Boolean drawIt,Boolean hasGrow,
                 Boolean scrollHoriz,Boolean scrollVert);
```

`rView` Rectangle in which to display the list. This rectangle is in the local coordinates of the window passed in the `theWindow` parameter, and does not include the area occupied by the list's scroll bars.

`dataBounds` Initial data bounds. For example, to create a list with 5 columns and 10 rows, set the `left` and `top` fields to (0,0) and the `right` and `bottom` fields to 5 and 10 respectively.

`cSize` Size of each cell. If your application is using the default list definition function and passes (0,0) in this field, the size is calculated automatically.

`theProc` Pass `0` in this parameter to cause the default list definition function to be used.

`theWindow` Reference to the window in which the list is to be installed.

`drawIt` Specifies whether automatic drawing mode is to be initially enabled. When automatic redrawing is enabled (by setting this parameter to `true`), the list is automatically redrawn whenever it is changed.

This setting can be changed later using `LSetDrawingMode`. If your application chooses to disable automatic drawing mode (for example, for aesthetic reasons while adding rows and columns to a list) it should do so only for short periods of time.

`hasGrow` Specifies whether space should be left for a size box/resize control.

`scrollHoriz` Pass `true` if your list requires a horizontal scroll bar, otherwise pass `false`.

`scrollVert` Pass `true` if your list requires a vertical scroll bar, otherwise pass `false`.

Creating Lists That Use a Custom List Definition Function

If you are creating a list that uses a custom list definition function, you should use the function `CreateCustomList` to create the list:

```
OSStatus  CreateCustomList(const Rect *rView,const ListBounds *dataBounds,
                           Point cellSize,const ListDefSpec *theSpec,
                           WindowRef theWindow,Boolean drawIt,Boolean hasGrow,
                           Boolean scrollHoriz,Boolean scrollVert,ListHandle *outList);
```

The main difference between `LNew` and `CreateCustomList` is that the former takes the resource ID of a list definition function whereas the latter takes a universal procedure pointer to a list definition function.

Carbon Note

In the Classic API, custom list definition functions are compiled separately as `'CODE'` resources and the resource ID is passed in the `theProc` parameter of `LNew`. In Carbon, custom definition functions (and, indeed, all other custom definition functions) cannot be stored in resources; accordingly, the function `CreateCustomList` was introduced with Carbon to accommodate that situation.

Drawing List Box Frame

The List Manager does not draw the list box frame around the list. Accordingly, this must be drawn by your application.

Drawing Focus Rectangle

In a window with multiple lists, you need to indicate to the user which list is the current list, that is, which list is the target of current mouse and keyboard activity. Accordingly, you should draw a focus rectangle around the current list. (See the list on the left in Fig 1). The focus rectangle should be removed when the window or dialog containing the lists is deactivated.

> **Note**
>
> A single list in a window should also be outlined with a focus rectangle if keyboard input could have some other effect in the window not related to the list (for example, if the list is in a dialog containing both a list and an edit text item).

Disposing of a List

When you are finished with a list, you should dispose of it using `LDispose`, which disposes of the list structure as well as the data associated with the list. `LDispose` does not, however, dispose of any application-specific data you might have stored in a relocatable block specified by the `userHandle` field of the list structure. This should be separately disposed of before the call to `LDispose`.

Adding Rows and Columns to a List

When an application creates a list, it might choose to, for example, pre-allocate the columns it needs and then add rows to the list one by one. It might also create the list and add both rows and columns to it later.

Rows are inserted into a list using `LAddRow` and deleted using `LDelRow`. Columns are inserted in a list using `LAddColumn` and deleted using `LDelColumn`.

Disabling and Enabling the Automatic Drawing Mode

`LSetDrawingMode` should be used to turn off the automatic drawing mode before making changes to a list. After the changes have been made, `LSetDrawingMode` should be called again, this time to turn the automatic drawing mode back on.

`InvalWindowRect` should be called after the second call to `LSetDrawingMode` to invalidate the rectangle containing the list and its scroll bars. `LUpdate`, which should be called when your application receives an update event, will then redraw the list.

Responding to Events in a List

Mouse-Down Events

As previously stated, when a mouse-down event occurs in a list, including in the associated scroll bar areas, your application must call `LClick`. If the click is outside the list's display rectangle or scroll bars, `LClick` returns immediately, otherwise it handles all user interaction until the user releases the mouse button. While the mouse button is down, the List Manager performs scrolling as necessary, selects or deselects cells as appropriate, and adjusts the scroll bars.

Note that `LClick` returns `true` if the click was a double click. If the list is in a dialog, your application should respond to a double click in the same way that it would respond to a click on the default (OK) button.

In the case of multiple lists, if the mouse-down occurs inside a non-current list's display rectangle or scroll bar area, your application should call its function for changing the current list.

Key-Down Events

If a key-down event is received, and assuming that your application supports cell selection by Arrow key and/or type selection, your application should call its appropriate functions. In the case of multiple lists, your application should also respond to Tab key presses by changing the current list.

Update Events

If an update event (Classic event model) or `kEventWindowDrawContent` event type (Carbon event model) is received, your application must call `LUpdate` to redraw the list. The region specified in the first parameter to the `LUpdate` call is usually the window's visible region as retrieved by `GetPortVisibleRegion`.

Your application will also need to draw the list box frame in the correct state (window active or inactive) and, if a focus rectangle is required and the window is active, the focus rectangle.

Activate Events

If an activate event (Classic event model) or `kEventWindowActivated` or `kEventWindowDeactivated` event type is received (Carbon event model), your application must call `LActivate` to activate or deactivate the list as appropriate. Your application will also need to draw the list box frame in the correct state, and either draw or erase the keyboard focus rectangle, depending on whether the window is becoming active or inactive.

If your application supports type selection in a list, it will also need to reset certain type selection variables when the window containing that list is becoming active.

Getting and Setting List Selections

The List Manager provides functions for determining which cells are currently selected and for selecting and deselecting cells. `LGetSelect` is used to either determine whether a specified cell is selected or to keep advancing from a specified starting cell until the next selected cell is found. `LSetSelect` is used to select or deselect a specified cell.

`LNextCell`, which simply advances from one cell in a list to the next, is often used in functions associated with getting and setting list selections.

Scrolling a List

LAutoScroll may be used to scroll the first selected cell to the upper-left corner of the list's display rectangle.

LScroll allows your application to scroll the list by a specified number of rows and/or columns. Typically, you would use LScroll when you want your application to scroll a list just enough so that a certain cell (such as the cell the user has just selected using the an Arrow key or type selection) is visible.

Storing Data

Your application can store data in a cell using LSetCell. LSetCell's parameters include a pointer to the data, the length of the data, the location of the cell whose data you wish to set, and a handle to the list containing the cell. The data stored in a cell might be sourced from, for example, a string list resource.

Adding to Data

Your application can append data to a cell using LAddToCell.

Getting Cell Data

LGetCell may be used to copy the contents of a cell into a buffer. LGetCellDataLocation may be used to obtain the address and length of a cell's data. Unlike LGetCell, LGetCellDataLocation does not make a copy of the data, and should thus be used when you want to access, but not manipulate, the data.

Clearing Data

Your application can remove all data from a cell using LClrCell.

Searching a List

Your application can use LSearch to search through a list for a particular item. LSearch takes, as one parameter, a universal procedure pointer to a **match function.** If NULL is specified for this parameter, LSearch searches the list for the first cell whose data matches the specified data, calling the Text Utilities IdenticalString function to compare each cell's data with the specified data until IdenticalString returns 0, indicating that a match has been found.

Custom Match Functions

The default match function is useful for text-only lists. Your application can use a different match function to facilitate searches in other types of lists as long as that function is defined just like IdenticalString.

A common custom match function is one which supports type selection in lists, that is, one which works like the default match function but which allows the cell data to be longer than the data being searched for. For example, a search for the string "be" would match a cell containing the string "Beams."

Changing the Current List

As previously stated, when a window or dialog contains multiple lists, your application should allow the user to change the current list by clicking in one of the non-current lists or by pressing the Tab key or Shift-Tab. In a window with more than two lists, Tab key presses should make the next list in a predetermined sequence the current list, and Shift-Tab should make the previous list in that sequence the current list. The predetermined sequence is best implemented using a **linked ring.**

Linked Ring

To create a linked ring, you can use the refCon field of each list structure. Assign the handle to the second list to the refCon field of the first list, assign the handle to the third list to the refCon field of the second list, and so on, until, to close the circle, the handle to the first list is assigned to the refCon field of the last list. Then, in response to a Tab key press in the current list, your application can ascertain the next list in the ring by looking at the current list's refCon field.

Responding to Shift-Tab is a little more complex. The following example function shows how this can be done:

```
ListHandle gCurrentListHdl;

void  doFindPreviousListInRing(void)
{
  ListHandle listHdl;

  listHdl = gCurrentListHdl;

  while((ListHandle) GetListRefCon(listHdl) != gCurrentList)
    listHdl = (ListHandle) GetListRefCon(listHdl);

  gCurrentListHdl = listHdl;
}
```

Customizing the Cell-Selection Algorithm

You can modify the algorithm the List Manager uses to select cells in response to mouse clicking and dragging by changing the value in the `selFlags` field of the list structure. (Recall that, by default, mouse clicks deselect all cells and select the current cell, Shift-click and Shift-drag extend the selection as a rectangular range, and Command-click and Command drag toggle selections according to the selection state of the initial cell.)

The bits in the `selFlags` field are represented by the following constants. Those constants, and the effect the values they represent have on the cell-selection algorithm, are as follows:

Constant	*Value*	*Effect*
`lOnlyOne`	`128`	Allow only one cell to be selected at any one time.
`lExtendDrag`	`64`	Allow the user to select a range of cells by clicking the first cell and dragging to the last cell without necessarily press ing the Shift or Command key. (Ordinarily, dragging in this manner results in only the last cell being selected.)
`lNoDisjoint`	`32`	Prevent discontiguous selections using the Command key, while still allowing the user to select a contiguous range of cells.
`lNoExtend`	`16`	Cause all previously selected cells to be deselected when the user Shift-clicks.
`lNoRect`	`8`	Disable the feature which allows the user to shrink a selection by Shift-clicking to select a range of cells and then dragging the cursor to a position within that range. (With this feature is disabled, all cells in the cursor's path during a Shift-drag become selected even if the user drags the cursor back over the cell.)
`lUseSense`	`4`	Allow the user to deselect a range of cells by Shift- dragging. (Ordinarily, Shift-dragging causes cells to become selected even if the first cell clicked is already selected.)
`lNoNilHilite`	`2`	Turn off the highlighting of cells which contain no data. (Note that this constant is somewhat different from the others in that it affects the display of a list, not the way that the List Manager selects items in response to a click.)

These constants are often used additively. For example, you could make the Shift key work just like the Command key using the following code:

```
SetListSelectionFlags(listHdl,lNoRect + lNoExtend + lUseSense);
```

If your application customizes the cell-selection algorithm in lists that allow multiple cell selection, it should make the nonstandard behavior clear to the user. Typically, this is done by displaying explanatory text above the list's display rectangle.

The List Box Control

The list box control reduces the programming effort involved in managing lists. Basically, this control frees your application from the requirement to provide its own functions for attending to mouse and keyboard interaction with the list (except for type selection). To create and manage lists using the list box control, you need to

- Provide a list box `'CNTL'` resource and a list box description (`'ldes'`) resource.
- Provide functions for storing data in the list's cells and, where required, for adding rows and/or columns.
- Provide a function to support type selection, if required.
- Modify the list's cell selection algorithm, if required.
- Provide a function which searches for, and returns the data in, the selected cell or cells.

The handle to the control is assigned to the `refCon` field of the list structure. This allows a custom list definition function to determine whether the control should be drawn in the activated or deactivated state by looking at the `contrlHilite` field of the control structure.

List Box Variants, Values, Constants, and Resources

Variant and Control Definition ID

The list box CDEF resource ID is 22. The two available variants and their control definition IDs are as follows:

Variant	*Var Code*	*Control Definition ID*
List box	`0`	`352 kControlListBoxProc`
Autosizing list box	`1`	`353 kControlListBoxAutoSizeProc`

Control Values

Control Value	*Content*
Initial	Resource ID of the `'ldes'` resource holding the list box information. Reset to `0` after creation. An initial value of `0` indicates not to read an `'ldes'` resource. (See The List Box Description Resource, below.)
Minimum	Reserved. Set to `0`.
Maximum	Reserved. Set to `0`.

Control Data Tag Constants

Control Data Tag Constant	*Meaning and Data Type Returned or Set*
`kControlListBoxListHandleTag`	Gets a handle to a list box. **Data type returned:** `ListHandle`
`kControlListBoxDoubleClickTag`	Checks to see whether the most recent click in a list box was a double click. **Data type returned:** `Boolean`. If `true`, the last `click` was a double click. If `false`, not.

`kControlListBoxKeyFilterTag`	Gets or sets a key filter function. **Data type returned or set:** `ControlKeyFilterUPP`
`kControlListBoxFontStyleTag`	Gets or sets the font style. **Data type returned or set:** `ControlFontStyleRec`

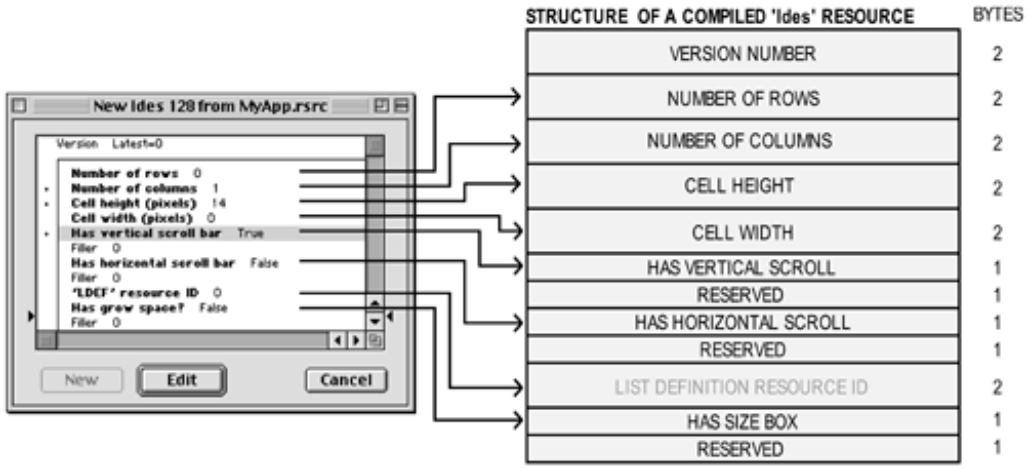
Control Part Codes

Constant	*Value*	*Description*
`kControlListBoxPart`	`24`	Event occurred in a list box.
`kControlListBoxDoubleClickPart`	25	Double-click occurred in a list box.

The List Box Description Resource

The list box description (`'ldes'`) resource, which must have resource ID of greater than 127, is used to specify information for a list box. The information is used by The Control Manager to provide additional information to the relevant list box control. Fig 3 shows the structure of a compiled `'ldes'` resource and such a resource being created using Resorcerer.

Figure 3. CREATING AN 'Ides' RESOURCE USING RESORCERER



The following describes the main fields of a compiled `'ldes'` resource:

Field	*Description*
`NUMBER OF ROWS`	The number of rows in the list box.
`NUMBER OF COLUMNS`	The number of columns in the list box.
`CELL HEIGHT`	The height of the list cells. Specify `0` to cause the height to be calculated automatically.
`CELL WIDTH`	The width of a the list cells. Specify `0` to cause the width to be calculated automatically.
`HAS VERTICAL SCROLL BAR`	`true` causes the list box to contain a vertical scroll bar.
`HAS HORIZONTAL SCROLL BAR`	`true` causes the list box to contain a horizontal scroll bar.
`RESOURCE ID`	The resource ID of the list definition function to use for the list. In Carbon, always set to `0`.

`HAS SIZE BOX`	`true` causes the List Manager to leave room for, and draw, a size box.

Programmatic Creation

List box controls may be created programmatically using the function `CreateListBoxControl`:

```
OSStatus  CreateListBoxControl(WindowRef window,const Rect *boundsRect,Boolean autoSize,
          SInt16 numRows,SInt16 numColumns,Boolean horizScroll,Boolean vertScroll,
          SInt16 cellHeight,SInt16 cellWidth,Boolean hasGrowSpace,
          const ListDefSpec *listDef,ControlRef *outControl);
```

Custom List Definition Functions

As previously stated, the default list definition function supports the display of unstyled text only. If your application needs to display items graphically, or display more than one type of information in each cell, you must create your own list definition (callback) function.

> **Note**
>
> For example, the MAC OS 8/9 Finder's About This Computer dialog contains a single-column list of applications currently in use. Each cell in the list contains an icon, the name of the application, the amount of memory in the application partition, and a graphical indication of how much of that memory has been used.

Your custom list definition (callback) function must be declared like this:

```
void  myListDefinition(SInt16 lMessage,Boolean lSelect,Rect *lRect,Cell lCell,
                       SInt16 lDataOffset,SInt16 lDataLen,ListHandle lHandle);
```

Messages Sent by List Manager

In essence, the sole requirement of your list definition function is to respond appropriately to four types of messages sent to it by the List Manager, and which are received in the `lMessage` parameter. The following constants define the four message types:

Constant	*Value*	*Meaning*
`lInitMsg`	`0`	Do any special list initialization.
`lDrawMsg`	`1`	Draw the cell.
`lHiliteMsg`	`2`	Invert the cell's highlight state.
`lCloseMsg`	`3`	Take any special disposal action.

The `lSelect`, `lRect`, `lCell`, `lDataOffset`, and `lDataLen` parameters, which contain information about the cell affected by the message, pass information to your custom list definition function only when the `lDrawMsg` or `lHiliteMsg` messages are received. The `lSelect` parameter indicates whether the cell should be highlighted. `lRect` and `lCell` provide the cell's rectangle and coordinates. `lDataOffset` and `lDataLen` parameters provide the offset and length of the cell's data referenced by the `cells` field of the list structure.

The Initialization Message

In response to the `lInitMsg` message, your application might, for example, change the `cellSize` and `indent` fields of the list structure. However, many custom list definition functions do not need to perform any action in response to the initialization message.

The Draw Message

In response to the `lDrawMsg` message, your custom list definition function must examine the specified cell's data and draw the cell as appropriate, ensuring that the characteristics of the drawing environment are not altered.

The HighLighting Message

In response to the `lHiliteMsg` message, your custom list definition function should highlight the cell's rectangle. The following example shows how this might be done:

```
void  doLDEFHighlight(Rect *cellRect)
{
  UInt8 hiliteVal;

  hiliteVal = LMGetHiliteMode();
  BitClr(&hiliteVal,pHiliteBit);
  LMSetHiliteMode(hiliteVal);

  InvertRect(cellRect);
}
```

Responding to the Close Message

The `lCloseMsg` is sent immediately before the List Manager disposes of the memory occupied by the list. Your custom list definition function needs to respond only if it needs to perform some special processing at that point, such as releasing any additional memory associated with the list.

Main List Manager Constants, Data Types, and Functions

Constants

Masks for listFlags Field of List Structure

```
lDoVAutoscroll      = 2     Allow vertical autoscrolling.
lDoHAutoscroll      = 1     Allow horizontal autoscrolling.
```

Masks for selFlags Field of List Structure

```
lOnlyOne            = -128  Allow only one item to be selected at once.
lExtendDrag         = 64    Enable multiple item selection without Shift.
lNoDisjoint         = 32    Prevent discontiguous selections.
lNoExtend           = 16    Reset list before responding to Shift-click.
lNoRect             = 8     Shift-drag selects items passed by cursor.
lUseSense           = 4     Allow use of Shift key to deselect items.
lNoNilHilite        = 2     Disable highlighting of empty cells.
```

Messages to List Definition Function

```
lInitMsg            = 0      Do any special list initialisation.
lDrawMsg            = 1      Draw the cell.
lHiliteMsg          = 2      Invert cell's highlight state.
lCloseMsg           = 3      Take any special disposal action.
```

Control Kind

```
kControlKindListBox = FOUR_CHAR_CODE('lbox')
```

Data Types

```
typedef Point   Cell;
typedef Rect    ListBounds;
typedef char    DataArray[32001];
typedef char    *DataPtr;
typedef DataPtr *DataHandle;
```

List Structure

```
struct ListRec
{
  Rect             rView;         // List's display rectangle.
  GrafPtr          port;          // List's graphics port.
  Point            indent;        // Indent distance for drawing.
  Point            cellSize;      // Size in pixels of a cell.
  Rect             visible;       // Boundary of visible cells.
  ControlRef       vScroll;       // Vertical scroll bar.
  ControlRef       hScroll;       // Horizontal scroll bar.
  SInt8            selFlags;      // Selection flags.
  Boolean          lActive;       // true if list is active.
  SInt8            lReserved;     // (Reserved.)
  Sint8            listFlags;     // Automatic scrolling flags.
  long             clikTime;      // TickCount at time of last tick.
  Point            clikLoc;       // Position of last click.
  Point            mouseLoc;      // Current mouse location.
  ListClickLoopUPP lClickLoop;    // Function called by LClick.
  Cell             lastClick;     // Last cell clicked.
  long             refCon;        // For application use.
```

```
  Handle           listDefProc;   // List definition function.
  Handle           userHandle;    // For application use.
  ListBounds       dataBounds;    // Boundary of cells allocated.
  DataHandle       cells;         // Cell data.
  short            maxIndex;      // (Used internally.)
  short            cellArray[1];  // Offsets to data.
};

typedef struct ListRec ListRec;
typedef ListRec *ListPtr;
typedef ListPtr *ListHandle;
```

Functions

Creating and Disposing of Lists

```
ListHandle       LNew(const Rect *rView,const ListBounds *dataBounds,Point cSize,
                 short theProc,WindowRef theWindow,Boolean drawIt,Boolean hasGrow,
                 Boolean scrollHoriz,Boolean scrollVert);
OSStatus         CreateCustomList(const Rect *rView,const ListBounds *dataBounds,
                 Point cellSize,const ListDefSpec *theSpec,WindowRef theWindow,Boolean
drawIt,
                 Boolean hasGrow,Boolean scrollHoriz,Boolean scrollVert,ListHandle
*outList);
void             LDispose(ListHandle lHandle);
```

Creating List Box Controls

```
OSStatus         CreateListBoxControl(WindowRef window,const Rect *boundsRect,
                 Boolean autoSize,SInt16 numRows,SInt16 numColumns,Boolean horizScroll,
                 Boolean vertScroll,SInt16 cellHeight,SInt16 cellWidth,Boolean hasGrowSpace,
                 const ListDefSpec *listDef,ControlRef *outControl);
```

Adding and Deleting Rows and Columns

```
short            LAddColumn(short count,short colNum,ListHandle lHandle);
short            LAddRow(short count,short rowNum,ListHandle lHandle);
void             LDelColumn(short count,short colNum,ListHandle lHandle);
void             LDelRow(short count,short rowNum,ListHandle lHandle);
```

Determining or Changing a Selection

```
Boolean          LGetSelect(Boolean next,Cell *theCell,ListHandle lHandle);
void             LSetSelect(Boolean setIt,Cell theCell,ListHandle lHandle);
```

Accessing and Manipulating Data Cells

```
void             LSetCell(const void *dataPtr,short dataLen,Cell theCell,ListHandle
lHandle);
void             LAddToCell(const void *dataPtr,short dataLen,Cell theCell,
                 ListHandle lHandle);
void             LClrCell(Cell theCell,ListHandle lHandle);
void             LGetCellDataLocation(short *offset,short *len,Cell theCell,
                 ListHandle lHandle);
void             LGetCell(void *dataPtr,short *dataLen,Cell theCell,ListHandle lHandle);
```

Responding to Events

```
Boolean          LClick(Point pt,short modifiers,ListHandle lHandle);
void             LUpdate(RgnHandle theRgn,ListHandle lHandle);
void             LActivate(Boolean act,ListHandle lHandle);
```

Modifying a List's Appearance

```
void            LSetDrawingMode(Boolean drawIt,ListHandle lHandle);
void            LDraw(Cell theCell,ListHandle lHandle);
void            LAutoScroll(ListHandle lHandle);
void            LScroll(short dCols,short dRows,ListHandle lHandle);
```

Searching for a List Containing a Particular Item

```
Boolean         LSearch(const void *dataPtr,short dataLen,ListSearchUPP searchProc,
                Cell *theCell,ListHandle lHandle);
```

Changing the Size of Cells and Lists

```
void            LSize(short listWidth,short listHeight,ListHandle lHandle);
void            LCellSize(Point cSize,ListHandle lHandle);
```

Getting Information About Cells

```
Boolean         LNextCell(Boolean hNext,Boolean vNext,Cell *theCell,ListHandle lHandle);
void            LRect(Rect *cellRect,Cell theCell,ListHandle lHandle);
Cell            LLastClick(ListHandle lHandle);
```

List Structure Accessor Functions

```
Rect*           GetListViewBounds(ListRef list,Rect *view);
void            SetListViewBounds(ListRef list,const Rect *view);
CGrafPtr        GetListPort(ListRef list);
void            SetListPort(ListRef list,CGrafPtr port);
Point*          GetListCellIndent(ListRef list,Point *indent);
void            SetListCellIndent(ListRef list,Point *indent);
Point*          GetListCellSize(ListRef list,Point *size);
void            SetListCellSize(ListRef list,Point *size);
ListBounds *    GetListVisibleCells(ListRef list,ListBounds *visible);
void            SetListVisibleCells(ListRef list,ListBounds *visible);
ControlHandle   GetListVerticalScrollBar(ListRef list);
void            SetListVerticalScrollBar(ListRef list,ControlHandle vScroll);
ControlHandle   GetListHorizontalScrollBar(ListRef list);
void            SetListHorizontalScrollBar(ListRef list,ControlHandle hScroll);
OptionBits      GetListSelectionFlags(ListRef list);
void            SetListSelectionFlags(ListRef list,OptionBits selectionFlags);
Boolean         GetListActive(ListRef list);
OptionBits      GetListFlags(ListRef list);
void            SetListFlags(ListRef list,OptionBits listFlags);
SInt32          GetListClickTime(ListRef list);
void            SetListClickTime(ListRef list,SInt32 time);
Point*          GetListClickLocation(ListRef list,Point *click);
Point*          GetListMouseLocation(ListRef list,Point *mouse);
ListClickLoopUPP GetListClickLoop(ListRef list);
void            SetListClickLoop(ListRef list,ListClickLoopUPP clickLoop);
void            SetListLastClick(ListRef list,Cell *lastClick);
SInt32          GetListRefCon(ListRef list);
void            SetListRefCon(ListRef list,SInt32 refCon);
Handle          GetListUserHandle(ListRef list);
void            SetListUserHandle(ListRef list,Handle userHandle);
Handle          GetListDefinition(ListRef list);
void            SetListDefinition(ListRef list,Handle listDefProc);
ListBounds*     GetListDataBounds(ListRef list,ListBounds *bounds);
DataHandle      GetListDataHandle(ListRef list);
```

Creating and Disposing of Universal Procedure Pointers

```
ListSearchUPP   NewListSearchUPP(ListSearchProcPtr userRoutine);
ListClickLoopUPP NewListClickLoopUPP(ListClickLoopProcPtruserRoutine);
```

```
ListDefUPP        NewListDefUPP(ListDefProcPtr userRoutine);
void              DisposeListSearchUPP(ListSearchUPP userUPP);
void              DisposeListClickLoopUPP(ListClickLoopUPP userUPP);
void              DisposeListDefUPP(ListDefUPP userUPP);
```

Application-Defined (Callback) Function

```
void              myListDefinition(short lMessage, Boolean lSelect, Rect *lRect, Cell lCell,
                  short lDataOffset, short lDataLen, ListHandle lHandle);
```

Relevant Text Utilities Data Type and Functions

Data Type

```
struct TypeSelectRecord
{
  unsigned long tsrLastKeyTime;
  ScriptCode    tsrScript;
  Str63         tsrKeyStrokes;
};
typedef struct TypeSelectRecord TypeSelectRecord;
```

Functions

```
void      TypeSelectClear(TypeSelectRecord *tsr);
Boolean   TypeSelectNewKey  (const EventRecord *theEvent,TypeSelectRecord *tsr);
```

Demonstration Program Lists Listing

```
// ****************************************************************************************
// Lists.h                                                                  CARBON EVENT MODEL
// ****************************************************************************************
//
// This program allows the user to open a window and a movable modal dialog by choosing the
// relevant items in the Demonstration menu. The window and the dialog both contain two
// lists.
//
// The cells of one list in the window, and of both lists in the dialog, contain text. The
// cells of the second list in the window contain icons.
//
// The text lists use the default list definition function. The list with the icons uses a
// custom list definition function.
//
// The currently active list is indicated by a keyboard focus frame, and can be changed by
// clicking in the non-active list or by pressing the tab key.
//
// The text list in the window uses the default cell-selection algorithm; accordingly,
// multiple cells, including discontiguous multiple cells, may be selected. The cell-
// selection algorithm for the other lists is customised so as to allow the selection of only
// one cell at a time.
//
// All lists support arrow key selection. The text list in the window and one of the lists in
// the dialog support type selection.
//
// The window is provided with an "Extract" push button. When this button is clicked, or when
// the user double clicks in one of the lists, the current selections in the lists are
// extracted and displayed in the bottom half of the window. In the dialog, the user's
// selections are displayed in static text fields embedded in placards below each list.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  An 'MBAR' resource, and 'MENU' resources for Apple, File, Edit and Demonstration menus
//    (preload, non-purgeable).
//
// •  A'DLOG' resource (purgeable) (initially not visible) and associated 'dlgx', 'dftb', and
//    'DITL' resources (purgeable).
//
// •  'CNTL' resources (purgeable) for various controls in both the window and dialog box,
//    including the list controls for the dialog box.
//
// •  'ldes' resources associated with the list controls for the dialog box.
//
// •  'STR#' resources (purgeable) containing the text strings for the text lists and for the
//    titles of the icons.
//
// •  An icon suite (non-purgeable) containing the icons for icon list.
//
// •  An 'LDEF' resource (preload, locked, non-purgeable) containing the custom list
//    definition function  used by the icon list.
//
// •  'hrct' and 'hwin' (purgeable) resources for balloon help.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// ****************************************************************************************

// ..................................................................... includes
```

```
#include <Carbon.h>

// ..................................................................................... defines

#define rMenubar                128
#define mAppleApplication       128
#define  iAbout                 1
#define mFile                   129
#define  iQuit                  12
#define mDemonstration          131
#define  iHandMadeLists         1
#define  iListControlLists      2
#define cExtractButton          128
#define cGroupBox               129
#define cSoftwareStaticText     130
#define cHardwareStaticText     131
#define rTextListStrings        128
#define rIconListIconSuiteBase  128
#define rIconListStrings        129
#define rListsDialog            128
#define  iDateFormatList        4
#define  iWatermarkList         5
#define  iDateFormatStaticText  7
#define  iWatermarkStaticText   9
#define  rDateFormatStrings     130
#define  rWatermarkStrings      131
#define kUpArrow                0x1e
#define kDownArrow              0x1f
#define kTab                    0x09
#define kScrollBarWidth         15
#define kMaxKeyThresh           120
#define kSystemLDEF             0

// ..........................................................................................
typedefs

typedef struct
{
  ListHandle textListHdl;
  ListHandle iconListHdl;
  ControlRef extractButtonHdl;
}  docStructure, **docStructureHandle;

typedef struct
{
  RGBColor     backColor;
  PixPatHandle backPixelPattern;
  Pattern      backBitPattern;
}  backColorPattern;

// ..................................................................................... function
prototypes

void        main                        (void);
void        doPreliminaries             (void);
OSStatus    appEventHandler             (EventHandlerCallRef,EventRef,void *);
OSStatus    windowEventHandler          (EventHandlerCallRef,EventRef,void *);
void        doAdjustMenus               (void);
void        doMenuChoice                (MenuID,MenuItemIndex);
void        doSaveBackground            (backColorPattern *);
void        doRestoreBackground         (backColorPattern *);
void        doSetBackgroundWhite        (void);
void        doOpenListsWindow           (void);
void        doKeyDown                   (SInt8,EventRecord *);
void        doDrawContent               (WindowRef);
void        doActivateDeactivate        (WindowRef,Boolean);
```

```
void        doInContent                (Point,UInt32);
void        doControlHit               (WindowRef,ControlRef,Point);
ListHandle  doCreateTextList           (WindowRef,Rect,SInt16,SInt16);
void        doAddRowsAndDataToTextList (ListHandle,SInt16,SInt16);
void        doAddTextItemAlphabetically (ListHandle,Str255);
ListHandle  doCreateIconList           (WindowRef,Rect,SInt16,ListDefUPP);
void        doAddRowsAndDataToIconList (ListHandle,SInt16);
void        doHandleArrowKey           (SInt8,EventRecord *,Boolean);
void        doArrowKeyMoveSelection    (ListHandle,SInt8,Boolean);
void        doArrowKeyExtendSelection  (ListHandle,SInt8,Boolean);
void        doTypeSelectSearch         (ListHandle,EventRecord *);
SInt16      searchPartialMatch         (Ptr,Ptr,SInt16,SInt16);
Boolean     doFindFirstSelectedCell    (ListHandle,Cell *);
void        doFindLastSelectedCell     (ListHandle,Cell *);
void        doFindNewCellLoc           (ListHandle,Cell,Cell *,SInt8,Boolean);
void        doSelectOneCell            (ListHandle,Cell);
void        doMakeCellVisible          (ListHandle,Cell);
void        doResetTypeSelection       (void);
void        doRotateCurrentList        (void);
void        doDrawFrameAndFocus        (ListHandle,Boolean);
void        doExtractSelections        (void);
void        doDrawSelections           (Boolean);
void        listDefFunction            (SInt16,Boolean,Rect *,Cell,SInt16,SInt16,ListHandle);
void        doLDEFDraw                 (Boolean,Rect *,Cell,SInt16,ListHandle);
void        doLDEFHighlight            (Rect *);

void        doListsDialog              (void);
Boolean     eventFilter                (DialogPtr,EventRecord *,SInt16 *);

// ********************************************************************************************
// Lists.c
// ********************************************************************************************

// ..................................................................................... includes

#include "Lists.h"

// ................................................................... global variables

ListDefUPP        gListDefFunctionUPP;
Boolean           gRunningOnX = false;
backColorPattern gBackColorPattern;

// ********************************************************************************************* main

void  main(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
EventTypeSpec applicationEvents[] = { { kEventClassApplication, kEventAppActivated    } ,
                                      { kEventClassCommand,     kEventProcessCommand  } ,
                                      { kEventClassMenu,        kEventMenuEnableItems } };

// ..................................................................... do preliminaries

  doPreliminaries();

// ..................................................... create universal procedure pointers

gListDefFunctionUPP = NewListDefUPP((ListDefProcPtr) listDefFunction);

// ..................................................................... set up menu bar and menus

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    ExitToShell();
```

```
  SetMenuBar(menubarHdl);
  DrawMenuBar();

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
      DisableMenuItem(menuRef,0);
    }

    gRunningOnX = true;
  }
  else
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
      SetMenuItemCommandID(menuRef,iQuit,kHICommandQuit);
  }

// ..................................................... install application event handler

  InstallApplicationEventHandler(NewEventHandlerUPP((EventHandlerProcPtr) appEventHandler),
                                 GetEventTypeCount(applicationEvents),applicationEvents,
                                 0,NULL);

  // ........................................................... run application event loop

  RunApplicationEventLoop();
}
// ********************************************************************************* doPreliminaries

void  doPreliminaries(void)
{
  MoreMasterPointers(256);
  InitCursor();
}

// ************************************************************************************ appEventHandler

OSStatus  appEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                          void * userData)
{
  OSStatus      result = eventNotHandledErr;
  UInt32        eventClass;
  UInt32        eventKind;
  HICommand     hiCommand;
  MenuID        menuID;
  MenuItemIndex menuItem;
  WindowClass   windowClass;

  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);

  switch(eventClass)
  {
    case kEventClassApplication:
      if(eventKind == kEventAppActivated)
        SetThemeCursor(kThemeArrowCursor);
      break;

    case kEventClassCommand:
      if(eventKind == kEventProcessCommand)
      {
```

```
        GetEventParameter(eventRef,kEventParamDirectObject,typeHICommand,NULL,
                          sizeof(HICommand),NULL,&hiCommand);
        menuID = GetMenuID(hiCommand.menu.menuRef);
        menuItem = hiCommand.menu.menuItemIndex;
        if((hiCommand.commandID != kHICommandQuit) &&
           (menuID >= mAppleApplication && menuID <= mDemonstration))
        {
          doMenuChoice(menuID,menuItem);
          result = noErr;
        }
      }
      break;

      case kEventClassMenu:
      if(eventKind == kEventMenuEnableItems)
      {
        GetWindowClass(FrontWindow(),&windowClass);
        if(windowClass == kDocumentWindowClass)
          doAdjustMenus();
        result = noErr;
      }
      break;
  }

  return result;
}

// ********************************************************************** windowEventHandler

OSStatus  windowEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                             void* userData)
{
  OSStatus           result = eventNotHandledErr;
  UInt32             eventClass;
  UInt32             eventKind;
  WindowRef          windowRef;
  EventRecord        eventRecord;
  docStructureHandle docStrucHdl;
  ControlRef         controlRef = NULL;
  ControlPartCode    controlPartCode;
  SInt8              charCode;
  UInt32             modifiers;
  Point              mouseLocation;

  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);

  switch(eventClass)
  {
    case kEventClassWindow:                                          // event class window
      GetEventParameter(eventRef,kEventParamDirectObject,typeWindowRef,NULL,sizeof(windowRef),
                        NULL,&windowRef);
      switch(eventKind)
      {
        case kEventWindowDrawContent:
          doDrawContent(windowRef);
          result = noErr;
          break;

        case kEventWindowActivated:
          doActivateDeactivate(windowRef,true);
          result = noErr;
          break;

        case kEventWindowDeactivated:
          doActivateDeactivate(windowRef,false);
          result = noErr;
```

```
          break;

        case kEventWindowClickContentRgn:
          GetEventParameter(eventRef,kEventParamMouseLocation,typeQDPoint,NULL,
                            sizeof(mouseLocation),NULL,&mouseLocation);
          SetPortWindowPort(FrontWindow());
          GlobalToLocal(&mouseLocation);
          GetEventParameter(eventRef,kEventParamKeyModifiers,typeUInt32,NULL,
                            sizeof(modifiers),NULL,&modifiers);
          doInContent(mouseLocation,modifiers);
          result = noErr;
          break;

        case kEventWindowClose:
          docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
          LDispose((*docStrucHdl)->textListHdl);
          LDispose((*docStrucHdl)->iconListHdl);
          DisposeHandle((Handle) docStrucHdl);
          DisposeWindow(windowRef);
          result = noErr;
          break;
      }
      break;

    case kEventClassMouse:                                          // event class mouse
      switch(eventKind)
      {
        case kEventMouseDown:
          GetEventParameter(eventRef,kEventParamMouseLocation,typeQDPoint,NULL,
                            sizeof(mouseLocation),NULL,&mouseLocation);
          SetPortWindowPort(FrontWindow());
          GlobalToLocal(&mouseLocation);
          controlRef = FindControlUnderMouse(mouseLocation,FrontWindow(),&controlPartCode);
          if(controlRef)
          {
            doControlHit(FrontWindow(),controlRef,mouseLocation);
            result = noErr;
          }
          break;
      }
      break;

    case kEventClassKeyboard:                                       // event class keyboard
      switch(eventKind)
      {
        case kEventRawKeyDown:
          GetEventParameter(eventRef,kEventParamKeyMacCharCodes,typeChar,NULL,
                            sizeof(charCode),NULL,&charCode);
          GetEventParameter(eventRef,kEventParamKeyModifiers,typeUInt32,NULL,
                            sizeof(modifiers),NULL,&modifiers);
          eventRecord.what      = keyDown;
          eventRecord.message   = charCode;
          eventRecord.when      = EventTimeToTicks(GetEventTime(eventRef));
          eventRecord.modifiers = modifiers;
          doKeyDown(charCode,&eventRecord);
          result = noErr;
          break;
      }
      break;
  }
  return result;
}

// ********************************************************************************* doAdjustMenus

void  doAdjustMenus(void)
{
```

```
  MenuRef menuRef;

  menuRef = GetMenuRef(mDemonstration);

  if(FrontWindow())
    DisableMenuItem(menuRef,1);
  else
    EnableMenuItem(menuRef,1);
}

// ********************************************************************************* doMenuChoice

void  doMenuChoice(MenuID menuID,MenuItemIndex menuItem)
{
  if(menuID == 0)
    return;

  switch(menuID)
  {
    case mAppleApplication:
      if(menuItem == iAbout)
        SysBeep(10);
      break;

    case mDemonstration:
      if(menuItem == iHandMadeLists)
        doOpenListsWindow();
      else if(menuItem == iListControlLists)
        doListsDialog();
      break;
  }
}

// ********************************************************************************* doSaveBackground

void  doSaveBackground(backColorPattern *gBackColorPattern)
{
  GrafPtr      currentPort;
  PixPatHandle backPixPatHdl;

  GetPort(&currentPort);

  GetBackColor(&gBackColorPattern->backColor);
  gBackColorPattern->backPixelPattern  = NULL;

  backPixPatHdl = NewPixPat();
  GetPortBackPixPat(currentPort,backPixPatHdl);
  if((*backPixPatHdl)->patType != 0)
    gBackColorPattern->backPixelPattern = backPixPatHdl;
  else
    gBackColorPattern->backBitPattern = *(PatPtr) (*(*backPixPatHdl)->patData);

  DisposePixPat(backPixPatHdl);
}

// ******************************************************************************** doRestoreBackground

void  doRestoreBackground(backColorPattern *gBackColorPattern)
{
  RGBBackColor(&gBackColorPattern->backColor);

  if(gBackColorPattern->backPixelPattern)
    BackPixPat(gBackColorPattern->backPixelPattern);
  else
    BackPat(&gBackColorPattern->backBitPattern);
}
```

```
// ********************************************************************** doSetBackgroundWhite

void  doSetBackgroundWhite(void)
{
  RGBColor whiteColor = {  0xFFFF, 0xFFFF, 0xFFFF };
  Pattern  whitePattern;

  RGBBackColor(&whiteColor);
  GetQDGlobalsWhite(&whitePattern);
  BackPat(&whitePattern);
}

// ****************************************************************************************************
// WindowList.c
// ****************************************************************************************************

// ..................................................................................... includes

#include "Lists.h"

// ................................................................................ global variables

ListHandle       gCurrentListHdl;
Str255           gStringArray[16];
TypeSelectRecord gTSStruct;
SInt16           gTSResetThreshold;
ListHandle       gTSLastListHit;

extern ListDefUPP        gListDefFunctionUPP;
extern backColorPattern gBackColorPattern;
extern Boolean           gRunningOnX;
// ********************************************************************** doOpenListsWindow

void  doOpenListsWindow(void)
{
  OSStatus           osError;
  Rect               contentRect = {  100,100,502,357 };
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  ControlRef         controlRef;
  Str255             software = "\pSoftware:";
  Str255             hardware = "\pHardware:";
  SInt16             fontNum;
  Rect               textListRect, pictListRect;
  ListHandle         textListHdl, iconListHdl;
  EventTypeSpec   windowEvents[] = { { kEventClassWindow,   kEventWindowDrawContent     } ,
                                     { kEventClassWindow,   kEventWindowActivated       } ,
                                     { kEventClassWindow,   kEventWindowDeactivated     } ,
                                     { kEventClassWindow,   kEventWindowClickContentRgn } ,
                                     { kEventClassWindow,   kEventWindowClose           } ,
                                     { kEventClassMouse,    kEventMouseDown             } ,
                                     { kEventClassKeyboard, kEventRawKeyDown            } };

  // .................. open window, attach document structure, set background color/pattern

  osError = CreateNewWindow(kDocumentWindowClass,kWindowStandardHandlerAttribute,
                            &contentRect,&windowRef);
  if(osError != noErr)
    ExitToShell();

  ChangeWindowAttributes(windowRef,kWindowCloseBoxAttribute,0);
  RepositionWindow(windowRef,NULL,kWindowAlertPositionOnMainScreen);
  SetWTitle(windowRef,"\pHand Made Lists");

  if(!(docStrucHdl = (docStructureHandle) NewHandle(sizeof(docStructure))))
    ExitToShell();
  SetWRefCon(windowRef,(SInt32) docStrucHdl);
```

```
  SetPortWindowPort(windowRef);

  SetThemeWindowBackground(windowRef,kThemeBrushDialogBackgroundActive,true);
  if(!gRunningOnX)
    doSaveBackground(&gBackColorPattern);
  else
    doSetBackgroundWhite();

  // ..................................................................... create window's controls

  if(!gRunningOnX)
    CreateRootControl(windowRef,&controlRef);

  if(!((*docStrucHdl)->extractButtonHdl = GetNewControl(cExtractButton,windowRef)))
    ExitToShell();
  if(!(controlRef = GetNewControl(cSoftwareStaticText,windowRef)))
    ExitToShell();
  SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,software[0],
                 &software[1]);

  if(!(controlRef = GetNewControl(cHardwareStaticText,windowRef)))
    ExitToShell();
  SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,hardware[0],
                 &hardware[1]);

  if(!(controlRef = GetNewControl(cGroupBox,windowRef)))
    ExitToShell();

  // ..................................................................... set window's font

  GetFNum("\pGeneva",&fontNum);
  TextFont(fontNum);
  TextSize(10);

  // ............................... create lists, assign handles to document structure fields

  SetRect(&textListRect,21,26,151,130);
  SetRect(&pictListRect,169,26,236,130);

  textListHdl = doCreateTextList(windowRef,textListRect,1,kSystemLDEF);
  iconListHdl = doCreateIconList(windowRef,pictListRect,1,gListDefFunctionUPP);

  (*docStrucHdl)->textListHdl = textListHdl;
  (*docStrucHdl)->iconListHdl  = iconListHdl;

  // ........................... assign handles to list structure refCon fields for linked ring

  SetListRefCon(textListHdl,(SInt32) iconListHdl);
  SetListRefCon(iconListHdl,(SInt32) textListHdl);

  // ..................................................... make text list the current list

  gCurrentListHdl = textListHdl;

  // ..................................................... install window event handler

  InstallWindowEventHandler(windowRef,
                            NewEventHandlerUPP((EventHandlerProcPtr) windowEventHandler),
                            GetEventTypeCount(windowEvents),windowEvents,0,NULL);

  // ..................................... save and set background color/pattern, show window

  ShowWindow(windowRef);
}

// ********************************************************************************** doKeyDown
```

```
void  doKeyDown(SInt8 charCode,EventRecord * eventStrucPtr)
{
  docStructureHandle docStrucHdl;
  Boolean            allowExtendSelect;

  docStrucHdl = (docStructureHandle) GetWRefCon(FrontWindow());

  if(charCode == kTab)
  {
    doRotateCurrentList();
  }
  else if(charCode == kUpArrow || charCode == kDownArrow)
  {
    if(gCurrentListHdl == (*docStrucHdl)->textListHdl)
      allowExtendSelect = true;
    else
      allowExtendSelect = false;

    doHandleArrowKey(charCode,eventStrucPtr,allowExtendSelect);
  }
  else
  {
    if(gCurrentListHdl == (*docStrucHdl)->textListHdl)
      doTypeSelectSearch((*docStrucHdl)->textListHdl,eventStrucPtr);
  }
}

// ********************************************************************************* doDrawContent

void  doDrawContent(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  ListHandle         textListHdl, iconListHdl;
  RgnHandle          visibleRegionHdl = NewRgn();

  SetPortWindowPort(windowRef);

  GetPortVisibleRegion(GetWindowPort(windowRef),visibleRegionHdl);
  UpdateControls(windowRef,visibleRegionHdl);

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  textListHdl = (*docStrucHdl)->textListHdl;
  iconListHdl = (*docStrucHdl)->iconListHdl;

  if(visibleRegionHdl)
  {
    if(gRunningOnX)
    {
      if(IsWindowHilited(windowRef))
        TextMode(srcOr);
      else
        TextMode(grayishTextOr);
    }
    LUpdate(visibleRegionHdl,textListHdl);
    LUpdate(visibleRegionHdl,iconListHdl);
    DisposeRgn(visibleRegionHdl);
  }

  doDrawFrameAndFocus(textListHdl,IsWindowHilited(windowRef));
  doDrawFrameAndFocus(iconListHdl,IsWindowHilited(windowRef));

  doDrawSelections(windowRef == FrontWindow());
}

// ************************************************************************ doActivateDeactivate
```

```
void  doActivateDeactivate(WindowRef windowRef,Boolean becomingActive)
{
  GrafPtr            oldPort;
  ControlRef         controlRef;
  docStructureHandle docStrucHdl;
  ListHandle         textListHdl, iconListHdl;
  RgnHandle          visibleRegionHdl = NewRgn();

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

  if(!gRunningOnX)
  {
    GetRootControl(windowRef,&controlRef);
    if(becomingActive)
      ActivateControl(controlRef);
    else
      DeactivateControl(controlRef);
  }

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  textListHdl = (*docStrucHdl)->textListHdl;
  iconListHdl = (*docStrucHdl)->iconListHdl;

  if(visibleRegionHdl)
    GetPortVisibleRegion(GetWindowPort(windowRef),visibleRegionHdl);

  if(becomingActive)
  {
    LActivate(true,textListHdl);
    LActivate(true,iconListHdl);

    if(!gRunningOnX)
    {
      TextMode(srcOr);
      if(visibleRegionHdl)
      {
        LUpdate(visibleRegionHdl,textListHdl);
        LUpdate(visibleRegionHdl,iconListHdl);
        DisposeRgn(visibleRegionHdl);
      }
    }
    doDrawFrameAndFocus(textListHdl,true);
    doDrawFrameAndFocus(iconListHdl,true);

    doResetTypeSelection();

    doDrawSelections(true);
  }
  else
  {
    LActivate(false,textListHdl);
    LActivate(false,iconListHdl);

    if(!gRunningOnX)

    {
      TextMode(grayishTextOr);
      if(visibleRegionHdl)
      {
        LUpdate(visibleRegionHdl,textListHdl);
        LUpdate(visibleRegionHdl,iconListHdl);
        DisposeRgn(visibleRegionHdl);
      }
    }

    doDrawFrameAndFocus(textListHdl,false);
```

```
    doDrawFrameAndFocus(iconListHdl,false);

    doDrawSelections(false);
  }

  SetPort(oldPort);
}

// ********************************************************************************* doInContent

void  doInContent(Point mouseLocation,UInt32 modifiers)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  Rect               textListRect, pictListRect;
  Boolean            isDoubleClick;

  windowRef = FrontWindow();
  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

  GetListViewBounds((*docStrucHdl)->textListHdl,&textListRect);
  GetListViewBounds((*docStrucHdl)->iconListHdl,&pictListRect);

  if(PtInRect(mouseLocation,&textListRect) || PtInRect(mouseLocation,&pictListRect))
  {
    if((PtInRect(mouseLocation,&textListRect) &&
        gCurrentListHdl != (*docStrucHdl)->textListHdl) ||
       (PtInRect(mouseLocation,&pictListRect) &&
        gCurrentListHdl != (*docStrucHdl)->iconListHdl))
    {
      doRotateCurrentList();
    }
    isDoubleClick = LClick(mouseLocation,modifiers,gCurrentListHdl);
    if(isDoubleClick)
      doExtractSelections();
  }
}

// ********************************************************************************* doControlHit

void  doControlHit(WindowRef windowRef,ControlRef controlRef,Point mouseLocation)
{
  docStructureHandle docStrucHdl;

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

  if(controlRef == (*docStrucHdl)->extractButtonHdl)
  {
    if(TrackControl(controlRef,mouseLocation,NULL))
      doExtractSelections();
  }
  else
  {
    if((mouseLocation.h < 165 && gCurrentListHdl != (*docStrucHdl)->textListHdl) ||
       (mouseLocation.h > 165 && gCurrentListHdl != (*docStrucHdl)->iconListHdl))
    {
      doRotateCurrentList();
    }

    LClick(mouseLocation,0,gCurrentListHdl);
  }
}

// ********************************************************************************* doCreateTextList

ListHandle  doCreateTextList(WindowRef windowRef,Rect listRect,SInt16 numCols,SInt16 lDef)
{
```

```
  Rect       dataBounds;
  Point      cellSize;
  ListHandle textListHdl;
  Cell       theCell;

  SetRect(&dataBounds,0,0,numCols,0);
  SetPt(&cellSize,0,0);

  listRect.right = listRect.right - kScrollBarWidth;

  textListHdl = LNew(&listRect,&dataBounds,cellSize,lDef,windowRef,true,false,false,true);

  doAddRowsAndDataToTextList(textListHdl,rTextListStrings,15);

  SetPt(&theCell,0,0);
  LSetSelect(true,theCell,textListHdl);

  doResetTypeSelection();

  return textListHdl;
}

// ************************************************************** doAddRowsAndDataToTextList

void  doAddRowsAndDataToTextList(ListHandle textListHdl,SInt16 stringListID,
                                 SInt16 numberOfStrings)
{
  SInt16 stringIndex;
  Str255 theString;

  for(stringIndex = 1;stringIndex < numberOfStrings + 1;stringIndex++)
  {
    GetIndString(theString,stringListID,stringIndex);
    doAddTextItemAlphabetically(textListHdl,theString);
  }
}

// ************************************************************** doAddTextItemAlphabetically

void  doAddTextItemAlphabetically(ListHandle listHdl,Str255 theString)
{
  Boolean    found;
  ListBounds dataBounds;
  SInt16     totalRows, currentRow, cellDataOffset, cellDataLength;
  DataHandle dataHandle;
  Cell       aCell;

  found = false;

  GetListDataBounds(listHdl,&dataBounds);
  totalRows = dataBounds.bottom - dataBounds.top;
  currentRow = -1;

  while(!found)
  {
    currentRow += 1;
    if(currentRow == totalRows)
      found = true;
    else
    {
      SetPt(&aCell,0,currentRow);
      LGetCellDataLocation(&cellDataOffset,&cellDataLength,aCell,listHdl);

      dataHandle = GetListDataHandle(listHdl);
      MoveHHi((Handle) dataHandle);
      HLock((Handle) dataHandle);
```

```
      if(CompareText(theString + 1,((Ptr) (*listHdl)->cells[0] + cellDataOffset),
                     StrLength(theString),cellDataLength,NULL) == -1)
      {
        found = true;
      }

      HUnlock((Handle) dataHandle);
    }
  }
  currentRow = LAddRow(1,currentRow,listHdl);
  SetPt(&aCell,0,currentRow);

  LSetCell(theString + 1,(SInt16) StrLength(theString),aCell,listHdl);
}

// ************************************************************************** doCreateIconList

ListHandle  doCreateIconList(WindowRef windowRef,Rect listRect,SInt16 numCols,
                             ListDefUPP listDefinitionFunctionUPP)
{
  Rect        dataBounds;
  Point       cellSize;
  ListHandle  iconListHdl;
  Cell        theCell;
  ListDefSpec listDefSpec;

  SetRect(&dataBounds,0,0,numCols,0);
  SetPt(&cellSize,52,52);

  listRect.right = listRect.right - kScrollBarWidth;

  listDefSpec.u.userProc = listDefinitionFunctionUPP;

  CreateCustomList(&listRect,&dataBounds,cellSize,&listDefSpec,windowRef,true,false,false,
                   true,&iconListHdl);

  SetListSelectionFlags(iconListHdl,lOnlyOne);

  doAddRowsAndDataToIconList(iconListHdl,rIconListIconSuiteBase);

  SetPt(&theCell,0,0);
  LSetSelect(true,theCell,iconListHdl);

  return iconListHdl;
}

// ********************************************************************* doAddRowsAndDataToIconList

void  doAddRowsAndDataToIconList(ListHandle iconListHdl,SInt16 iconSuiteBase)
{
  ListBounds dataBounds;
  SInt16     rowNumber, suiteIndex, index = 0;
  Handle     iconSuiteHdl;
  Cell       theCell;

  GetListDataBounds(iconListHdl,&dataBounds);
  rowNumber = dataBounds.bottom;

  for(suiteIndex = iconSuiteBase;suiteIndex < (iconSuiteBase + 10);suiteIndex++)
  {
    GetIconSuite(&iconSuiteHdl,suiteIndex,kSelectorAllLargeData);
    rowNumber = LAddRow(1,rowNumber,iconListHdl);
    SetPt(&theCell,0,rowNumber);
    LSetCell(&iconSuiteHdl,sizeof(iconSuiteHdl),theCell,iconListHdl);

    rowNumber += 1;
  }
```

```
}

// ************************************************************************** doHandleArrowKey

void  doHandleArrowKey(SInt8 charCode,EventRecord * eventStrucPtr,Boolean allowExtendSelect)
{
  Boolean  moveToTopBottom = false;

  if(eventStrucPtr->modifiers & cmdKey)
    moveToTopBottom = true;

  if(allowExtendSelect && (eventStrucPtr->modifiers & shiftKey))
    doArrowKeyExtendSelection(gCurrentListHdl,charCode,moveToTopBottom);
  else
    doArrowKeyMoveSelection(gCurrentListHdl,charCode,moveToTopBottom);
}

// ************************************************************** doArrowKeyMoveSelection

void  doArrowKeyMoveSelection(ListHandle listHdl,SInt8 charCode,Boolean moveToTopBottom)
{
  Cell currentSelection, newSelection;

  if(doFindFirstSelectedCell(listHdl,&currentSelection))
  {
    if(charCode == kDownArrow)
      doFindLastSelectedCell(listHdl,&currentSelection);

    doFindNewCellLoc(listHdl,currentSelection,&newSelection,charCode,moveToTopBottom);

    doSelectOneCell(listHdl,newSelection);
    doMakeCellVisible(listHdl,newSelection);
  }
}

// ************************************************************** doArrowKeyExtendSelection

void  doArrowKeyExtendSelection(ListHandle listHdl,SInt8 charCode,Boolean moveToTopBottom)
{
  Cell currentSelection, newSelection;

  if(doFindFirstSelectedCell(listHdl,&currentSelection))
  {

    if(charCode == kDownArrow)
      doFindLastSelectedCell(listHdl,&currentSelection);

    doFindNewCellLoc(listHdl,currentSelection,&newSelection,charCode,moveToTopBottom);
    if(!(LGetSelect(false,&newSelection,listHdl)))
      LSetSelect(true,newSelection,listHdl);

    doMakeCellVisible(listHdl,newSelection);
  }
}

// ************************************************************************** doTypeSelectSearch

void  doTypeSelectSearch(ListHandle listHdl,EventRecord * eventStrucPtr)
{
  Cell          theCell;
  ListSearchUPP searchPartialMatchUPP;

  if((gTSLastListHit != listHdl) || ((eventStrucPtr->when - gTSStruct.tsrLastKeyTime) >=
     gTSResetThreshold) || (StrLength(gTSStruct.tsrKeyStrokes) == 63))
    doResetTypeSelection();

  gTSLastListHit = listHdl;
```

```
  gTSStruct.tsrLastKeyTime = eventStrucPtr->when;

  TypeSelectNewKey(eventStrucPtr,&gTSStruct);

  SetPt(&theCell,0,0);
  searchPartialMatchUPP  = NewListSearchUPP((ListSearchProcPtr) searchPartialMatch);

  if(LSearch(gTSStruct.tsrKeyStrokes + 1,StrLength(gTSStruct.tsrKeyStrokes),
             searchPartialMatchUPP,&theCell,listHdl))
  {
    LSetSelect(true,theCell,listHdl);
    doSelectOneCell(listHdl,theCell);
    doMakeCellVisible(listHdl,theCell);
  }

  DisposeListSearchUPP(searchPartialMatchUPP);
}

// ********************************************************************** searchPartialMatch

SInt16  searchPartialMatch(Ptr searchDataPtr,Ptr cellDataPtr,SInt16 cellDataLen,
                           SInt16 searchDataLen)
{
  SInt16 result;

  if((cellDataLen > 0) && (cellDataLen >= searchDataLen))
    result = IdenticalText(cellDataPtr,searchDataPtr,searchDataLen,searchDataLen,NULL);
  else
    result = 1;

  return result;
}

// ********************************************************************* doFindFirstSelectedCell

Boolean  doFindFirstSelectedCell(ListHandle listHdl,Cell *theCell)
{
  Boolean result;

  SetPt(theCell,0,0);
  result = LGetSelect(true,theCell,listHdl);

  return result;
}

// ********************************************************************** doFindLastSelectedCell

void  doFindLastSelectedCell(ListHandle listHdl,Cell *theCell)
{
  Cell    aCell;
  Boolean moreCellsInList;

  if(doFindFirstSelectedCell(listHdl,&aCell))
  {
    while(LGetSelect(true,&aCell,listHdl))
    {
      *theCell = aCell;
      moreCellsInList = LNextCell(true,true,&aCell,listHdl);
    }
  }
}

// ************************************************************************** doFindNewCellLoc

void  doFindNewCellLoc(ListHandle listHdl,Cell oldCellLoc,Cell *newCellLoc,SInt8 charCode,
                       Boolean moveToTopBottom)
{
```

```
  ListBounds dataBounds;
  SInt16     listRows;

  GetListDataBounds(listHdl,&dataBounds);
  listRows = dataBounds.bottom - dataBounds.top;
  *newCellLoc = oldCellLoc;

  if(moveToTopBottom)
  {
    if(charCode == kUpArrow)
      (*newCellLoc).v = 0;
    else if(charCode == kDownArrow)
      (*newCellLoc).v = listRows - 1;
  }
  else
  {
    if(charCode ==  kUpArrow)
    {
      if(oldCellLoc.v != 0)
        (*newCellLoc).v = oldCellLoc.v - 1;
    }
    else if(charCode == kDownArrow)
    {
      if(oldCellLoc.v != listRows - 1)
        (*newCellLoc).v = oldCellLoc.v + 1;
    }
  }
}

// ************************************************************************** doSelectOneCell

void  doSelectOneCell(ListHandle listHdl,Cell theCell)
{
  Cell    nextSelectedCell;
  Boolean moreCellsInList;

  if(doFindFirstSelectedCell(listHdl,&nextSelectedCell))
  {
    while(LGetSelect(true,&nextSelectedCell,listHdl))
    {
      if(nextSelectedCell.v != theCell.v)
        LSetSelect(false,nextSelectedCell,listHdl);
      else
        moreCellsInList = LNextCell(true,true,&nextSelectedCell,listHdl);
    }

    LSetSelect(true,theCell,listHdl);
  }
}

// ************************************************************************** doMakeCellVisible

void  doMakeCellVisible(ListHandle listHdl,Cell newSelection)
{
  ListBounds visibleRect;
  SInt16     dRows;

  GetListVisibleCells(listHdl,&visibleRect);

  if(!(PtInRect(newSelection,&visibleRect)))
  {
    if(newSelection.v > visibleRect.bottom - 1)
      dRows = newSelection.v - visibleRect.bottom + 1;
    else if(newSelection.v < visibleRect.top)
      dRows = newSelection.v - visibleRect.top;
    LScroll(0,dRows,listHdl);
  }
```

```
}

// ********************************************************************** doResetTypeSelection

void  doResetTypeSelection(void)
{
  TypeSelectClear(&gTSStruct);
  gTSLastListHit = NULL;
  gTSResetThreshold = 2 * LMGetKeyThresh();
  if(gTSResetThreshold > kMaxKeyThresh)
    gTSResetThreshold = kMaxKeyThresh;
}

// ********************************************************************** doRotateCurrentList

void  doRotateCurrentList(void)
{
  ListHandle oldListHdl, newListHdl;

  oldListHdl = gCurrentListHdl;

  newListHdl = (ListHandle) GetListRefCon(gCurrentListHdl);
  gCurrentListHdl = newListHdl;

  doDrawFrameAndFocus(oldListHdl,true);
  doDrawFrameAndFocus(newListHdl,true);
}

// ********************************************************************** doDrawFrameAndFocus

void  doDrawFrameAndFocus(ListHandle listHdl,Boolean inState)
{

  Rect borderRect;

  GetListViewBounds(listHdl,&borderRect);
  borderRect.right += kScrollBarWidth;

  if(!gRunningOnX)
    doRestoreBackground(&gBackColorPattern);
  else
    InvalWindowRect(FrontWindow(),&borderRect);

  DrawThemeFocusRect(&borderRect,false);

  if(inState)
    DrawThemeListBoxFrame(&borderRect,kThemeStateActive);
  else
    DrawThemeListBoxFrame(&borderRect,kThemeStateInactive);

  if(listHdl == gCurrentListHdl)
    DrawThemeFocusRect(&borderRect,inState);

  if(!gRunningOnX)
    doSetBackgroundWhite();
}

// ********************************************************************** doExtractSelections

void  doExtractSelections(void)
{
  docStructureHandle docStrucHdl;
  ListHandle         textListHdl, iconListHdl;
  SInt16             a, cellIndex, offset, dataLen;
  ListBounds         dataBounds;
  Cell               theCell;
  Rect               theRect;
```

```
  docStrucHdl = (docStructureHandle) GetWRefCon(FrontWindow());
  textListHdl = (*docStrucHdl)->textListHdl;
  iconListHdl = (*docStrucHdl)->iconListHdl;

  for(a=0;a<16;a++)
    gStringArray[a][0] = 0;

  GetListDataBounds(textListHdl,&dataBounds);

  for(cellIndex = 0;cellIndex < dataBounds.bottom;cellIndex++)
  {
    SetPt(&theCell,0,cellIndex);
    if(LGetSelect(false,&theCell,textListHdl))
    {
      LGetCellDataLocation(&offset,&dataLen,theCell,textListHdl);
      LGetCell(gStringArray[cellIndex] + 1,&dataLen,theCell,textListHdl);
      gStringArray[cellIndex][0] = (SInt8) dataLen;
    }
  }

  SetPt(&theCell,0,0);
  LGetSelect(true,&theCell,iconListHdl);
  GetIndString(gStringArray[15],rIconListStrings,theCell.v + 1);

  SetRect(&theRect,24,181,233,380);
  InvalWindowRect(FrontWindow(),&theRect);
}

// ********************************************************************** doDrawSelections

void  doDrawSelections(Boolean inState)
{
  Rect        theRect;
  SInt16      a, nextLine = 190;
  CFStringRef stringRef;

  if(inState == kThemeStateActive)
    TextMode(srcOr);
  else
    TextMode(grayishTextOr);

  SetRect(&theRect,22,179,235,382);
  EraseRect(&theRect);

  for(a=0;a<15;a++)
  {
    if(gStringArray[a][0] != 0)
    {
      stringRef = CFStringCreateWithPascalString(NULL,gStringArray[a],
                                                 kCFStringEncodingMacRoman);
      SetRect(&theRect,36,nextLine,240,nextLine + 15);
      DrawThemeTextBox(stringRef,kThemeSmallSystemFont,0,false,&theRect,teJustLeft,
                       NULL);
        nextLine += 12;
    }
  }

  stringRef = CFStringCreateWithPascalString(NULL,gStringArray[15],
                                             kCFStringEncodingMacRoman);

  SetRect(&theRect,170,190,240,205);
  DrawThemeTextBox(stringRef,kThemeSmallSystemFont,0,false,&theRect,teJustLeft,
                   NULL);

  TextMode(srcOr);
}
```

```
// ************************************************************************** listDefFunction

void  listDefFunction(SInt16 message,Boolean selected,Rect *cellRect,Cell theCell,
                      SInt16 dataOffset,SInt16 dataLen,ListHandle theList)
{
  switch(message)
  {
    case lDrawMsg:
      doLDEFDraw(selected,cellRect,theCell,dataLen,theList);
      break;
    case lHiliteMsg:
      doLDEFHighlight(cellRect);
      break;
  }
}

// ************************************************************************** doLDEFDraw

void  doLDEFDraw(Boolean selected,Rect *cellRect,Cell theCell,SInt16 dataLen,
                 ListHandle theList)
{
  GrafPtr oldPort;
  Rect    drawRect;
  Handle  iconSuiteHdl;
  Str255  theString;

  GetPort(&oldPort);

  SetPort(GetListPort(theList));

  EraseRect(cellRect);

  drawRect = *cellRect;

  drawRect.top += 2;
  drawRect.left += 10;
  drawRect.right -= 10;
  drawRect.bottom -= 18;
  if(dataLen == sizeof(Handle))
  {
    LGetCell(&iconSuiteHdl,&dataLen,theCell,theList);

    if(GetListActive(theList))
      PlotIconSuite(&drawRect,kAlignNone,kTransformNone,iconSuiteHdl);
    else
      PlotIconSuite(&drawRect,kAlignNone,kTransformDisabled,iconSuiteHdl);

  }

  GetIndString(theString,129,theCell.v + 1);
  SetRect(&drawRect,drawRect.left - 10,drawRect.top + 36,drawRect.right + 10,
          drawRect.bottom + 16);
  TETextBox(&theString[1],theString[0],&drawRect,teCenter);

  if(selected)
    doLDEFHighlight(cellRect);

  SetPort(oldPort);
}

// ************************************************************************** doLDEFHighlight

void  doLDEFHighlight(Rect *cellRect)
{
  UInt8 hiliteVal;
```

```
  hiliteVal = LMGetHiliteMode();
  BitClr(&hiliteVal,pHiliteBit);
  LMSetHiliteMode(hiliteVal);

  InvertRect(cellRect);
}

// ****************************************************************************************
// DialogLists.c                                                        CLASSIC EVENT MODEL
// ****************************************************************************************

// ..................................................................... includes

#include "Lists.h"

// ************************************************************************** doListsDialog

void  doListsDialog(void)
{
  DialogPtr      dialogPtr;
  GrafPtr        oldPort;
  ModalFilterUPP eventFilterUPP;
  ControlRef     dateFormatControlRef, watermarkControlRef, controlRef;
  ListHandle     dateFormatListHdl, watermarkListHdl;
  SInt16         itemHit;
  Cell           theCell;
  SInt16         dataLen, offset;
  Str255         dateFormatString, watermarkString;
  Boolean        wasDoubleClick = false;
  // ......................... explicitly deactivate front window if it exists, create dialog

  if(FrontWindow())
    doActivateDeactivate(FrontWindow(),false);

  if(!(dialogPtr = GetNewDialog(rListsDialog,NULL,(WindowRef) -1)))
    ExitToShell();

  GetPort(&oldPort);
  SetPortDialogPort(dialogPtr);

  // ............................................................... set default push button

  SetDialogDefaultItem(dialogPtr,kStdOkItemIndex);

  // ........................... create universal procedure pointer for event filter function

  eventFilterUPP = NewModalFilterUPP((ModalFilterProcPtr) eventFilter);

  // ........... add rows to lists, store data in their cells, modify cell selection algorithm

  GetDialogItemAsControl(dialogPtr,iDateFormatList,&dateFormatControlRef);
  GetControlData(dateFormatControlRef,kControlEntireControl,kControlListBoxListHandleTag,
                 sizeof(dateFormatListHdl),&dateFormatListHdl,NULL);

  doAddRowsAndDataToTextList(dateFormatListHdl,rDateFormatStrings,17);

  SetListSelectionFlags(dateFormatListHdl,lOnlyOne);

  SetPt(&theCell,0,0);
  LSetSelect(true,theCell,dateFormatListHdl);

  GetDialogItemAsControl(dialogPtr,iWatermarkList,&watermarkControlRef);
  GetControlData(watermarkControlRef,kControlEntireControl,kControlListBoxListHandleTag,
                 sizeof(watermarkListHdl),&watermarkListHdl,NULL);

  doAddRowsAndDataToTextList(watermarkListHdl,rWatermarkStrings,12);
```

```
  SetListSelectionFlags(watermarkListHdl,lOnlyOne);

  SetPt(&theCell,0,0);
  LSetSelect(true,theCell,watermarkListHdl);

  // ..................................................... show dialog and set keyboard focus

  ShowWindow(GetDialogWindow(dialogPtr));

  SetKeyboardFocus(GetDialogWindow(dialogPtr),dateFormatControlRef,1);

  // ........................................................... enter ModalDialog loop

  do
  {
    ModalDialog(eventFilterUPP,&itemHit);
    if(itemHit == iDateFormatList)
    {
      SetPt(&theCell,0,0);
      LGetSelect(true,&theCell,dateFormatListHdl);
      LGetCellDataLocation(&offset,&dataLen,theCell,dateFormatListHdl);
      LGetCell(dateFormatString + 1,&dataLen,theCell,dateFormatListHdl);
      dateFormatString[0] = (SInt8) dataLen;

      GetDialogItemAsControl(dialogPtr,iDateFormatStaticText,&controlRef);
      SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,
                     dateFormatString[0],&dateFormatString[1]);
      Draw1Control(controlRef);

      GetControlData(dateFormatControlRef,kControlEntireControl,
                     kControlListBoxDoubleClickTag,sizeof(wasDoubleClick),&wasDoubleClick,
                     NULL);
    }
    else if(itemHit == iWatermarkList)
    {
      SetPt(&theCell,0,0);
      LGetSelect(true,&theCell,watermarkListHdl);
      LGetCellDataLocation(&offset,&dataLen,theCell,watermarkListHdl);
      LGetCell(watermarkString + 1,&dataLen,theCell,watermarkListHdl);
      watermarkString[0] = (SInt8) dataLen;

      GetDialogItemAsControl(dialogPtr,iWatermarkStaticText,&controlRef);
      SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,
                     watermarkString[0],&watermarkString[1]);
      Draw1Control(controlRef);

      GetControlData(watermarkControlRef,kControlEntireControl,
                     kControlListBoxDoubleClickTag,sizeof(wasDoubleClick),&wasDoubleClick,
                     NULL);
    }
  }  while(itemHit != kStdOkItemIndex &&  wasDoubleClick == false);

  // ..................................................................... clean up

  DisposeDialog(dialogPtr);
  DisposeModalFilterUPP(eventFilterUPP);
  SetPort(oldPort);
}
0
// ********************************************************************** eventFilter

Boolean  eventFilter(DialogPtr dialogPtr,EventRecord *eventStrucPtr,SInt16 *itemHit)
{
  Boolean    handledEvent;
  GrafPtr    oldPort;
  SInt8      charCode;
  ControlRef controlRef, focusControlRef;
```

```
  ListHandle watermarkListHdl;
  handledEvent = false;
  GetPort(&oldPort);
  SetPortDialogPort(dialogPtr);

  if(eventStrucPtr->what == keyDown)
  {
    charCode = eventStrucPtr->message & charCodeMask;

    if(charCode != kUpArrow && charCode != kDownArrow && charCode != kTab)
    {
      GetDialogItemAsControl(dialogPtr,iWatermarkList,&controlRef);
      GetControlData(controlRef,kControlEntireControl,kControlListBoxListHandleTag,
                     sizeof(watermarkListHdl),&watermarkListHdl,NULL);
      GetKeyboardFocus(GetDialogWindow(dialogPtr),&focusControlRef);
      if(controlRef == focusControlRef)
      {
        doTypeSelectSearch(watermarkListHdl,eventStrucPtr);
        Draw1Control(controlRef);
      }

      handledEvent = true;
    }
  }
  else
  {
    handledEvent = StdFilterProc(dialogPtr,eventStrucPtr,itemHit);
  }

  SetPort(oldPort);
  return handledEvent;
}

// ********************************************************************************************
```

Demonstration Program Lists Comments

When this program is run, the user should open the window and movable modal dialog by choosing the relevant items in the Demonstration menu. The user should manipulate the lists in the window and dialog, noting their behavior in the following circumstances:

- Changing the active list (that is, the current target of mouse and keyboard activity) by clicking in the nonactive list and by using the Tab key to cycle between the two lists.
- Scrolling the active list using the vertical scroll bars, including dragging the scroll box/scroller and clicking in the scroll arrows and gray areas/track.
- Clicking, and clicking and dragging, in the active list so as to select a particular cell, including dragging the cursor above and below the list to automatically scroll the list to the desired cell.
- Pressing the Up-Arrow and Down-Arrow keys, noting that this action changes the selected cell and, where necessary, scrolls the list to make the newly selected cell visible.
- In the lists in the window
    - Double-clicking on a cell in the active list.
    - Pressing the Command-key as well as the Up-Arrow and Down-Arrow keys, noting that, in both the text list and the picture list, this results in the top-most or bottom-most cell being selected.
- In the "Software" list in the window
    - Shift-clicking and dragging in the list to make contiguous multiple cell selections.
    - Command-clicking and dragging in the list to make noncontiguous multiple cell selections, noting the differing effects depending on whether the cell initially clicked is selected or not selected.
    - Shift-clicking outside a block of multiple cell selections, including between two fairly widely separated noncontiguous selected cells.
    - Pressing the Shift-key as well as the Up-Arrow and Down-Arrow keys, noting that this results in multiple cell selections.
    - When the text list in the window, or the right hand list in the dialog, is the active list, typing the text of a particular cell so as to select that cell by type selection, noting the effects of any excessive delay between keystrokes.

The user should also send the program to the background and bring it to the foreground again, noting the list deactivation/activation effects.

Lists.h

defines

`rListsDialog` represents the resource ID of the dialog's `'DLOG'` resource. `CExtractButton` and the following three constants represent the resource IDs of the window's controls. The next three constants represent the resource IDs of resources containing the strings for the window's text list, the icon suite for the icon list, and the strings for the icon titles.

`rListsDialog` represents the resource ID of the dialog's `'DLOG'`, `'dlgx'`, and `'dftb'` resources. The following four constants represent the item numbers of items in the dialog's `'DITL'` resource. The next two constants represent the resource IDs of the `'STR#'` resources containing the strings for the dialog's lists.

The next three constants represent the character codes returned by the Up Arrow, Down Arrow, and Tab keys. `kScrollBarWidth` represents the width of the lists' vertical scroll bars. `kMaxKeyThresh` is used in the type selection function. `kSystemLDEF` represents the resource ID of the default list definition function.

typedefs

The type `docStructure` will be used to store the handles to the two list structures for the window and the reference to the window's push button. The handle to this structure will be stored in the window object.

The `backColorPattern` data type will be used to save and restore the background color and pattern.

Lists.c

`Lists.c` is simply the basic "engine" that supports the demonstration. There is little in this file that has not featured in previous demonstration programs.

main

A universal procedure pointer is created for the custom list definition function used by the second list in the window.

windowEventHandler

When the `kEventWindowClickContentRgn` event type is received, `doInContent` is called. The mouse location in local coordinates and the modifier keys are passed in the call.

When the kEventWindowClose event type is received, a handle to the window's document structure is retrieved so as to be able to pass the handles to the window's two list structures in the two calls to LDispose. LDispose disposes of all memory associated with the specified list.

When the kEventRawKeyDown event type is received, doKeyDown is called with the address of a variable of type EventRecord passed in the second parameter. Note that, in the case of this particular event type, the function ConvertEventRefToEventRecord does not "fill in" the event record's what, message, and modifiers fields. Accordingly, in lieu of a call to ConvertEventRefToEventRecord, the character code and modifiers are extracted from the event and a fully fleshed-out event record is constructed by the program prior to the call to doKeyDown.

doSaveBackground, doRestoreBackground, and doSetBackgroundWhite

doSaveBackground and doRestoreBackground save and restore the background color and the background bit or pixel pattern, and are called only when the program is run on Mac OS 8/9. doSetBackgroundWhite sets the background color to white and the background pattern to the pattern white.

WindowList.c

WindowList.c contains the functions pertaining to the lists in the window.

Global Variables

gCurrentListHandle will be assigned the handle to the list structure associated with the currently active list in the window. gStringArray will be assigned strings representing the selections from the lists. The next three global variables are associated with the type selection functions.

doOpenListsWindow

doOpenListsWindow creates the window and its controls, and calls the functions that create the two lists for the window.

SetThemeWindowBackground is called to set the window's background color/pattern and, if the program is running on Mac OS 8/9, doSaveBackground is called to save this background color/pattern for later use in the function doDrawFrameAndFocus. The call to doSetBackgroundWhite at this point is required only on Mac OS X to ensure that the background within the list frames is drawn in white.

`CreateRootControl` creates a root control for the window so as to simplify the task of activating and deactivating the window's controls. (This call is not required on Mac OS X because, on Mac OS X, the root control will be created automatically for windows which have at least one control.) The window's remaining controls are then created.

The calls to `doCreateTextList` and `doCreateIconList` create the lists. First, the rectangles in which the lists are to be displayed are defined. These are then passed in the calls to `doCreateTextList` and `doCreateIconList`. The handles to the list structures returned by these functions are then assigned to the relevant fields of the window's document structure.

The next block assigns the icon list's handle to the `refCon` field of the text list's list structure and the text list's handle to the `refCon` field of the icon list's list structure. This establishes the "linked ring" which will be used to facilitate the rotation of the active list via Tab key presses.

The next line establishes the text list as the currently active list.

doKeyDown

The first line gets the handle to the document structure.

If the key pressed was the Tab key, `doRotateCurrentList` is called to change the currently active list.

If the key pressed was either the Up Arrow or the Down Arrow key, and if the current list is the text list, a variable that specifies whether multiple cell selections via the keyboard are permitted is set to `true`. If the current list is the icon list, this variable is set to `false`. This variable is then passed as a parameter in a call to `doHandleArrowKey`, which further processes the Arrow key event.

If the key pressed was neither the Tab key, the Up Arrow key, or the Down Arrow key, and if the active list is the text list, the event is passed to `doTypeSelectSearch` (the type selection function) for further processing.

doDrawContent

`doDrawContent` is called when the `kEventWindowDrawContent` event type is received. The calls to `LUpdate` update (that is, redraw) the lists and the calls to `doDrawFrameAndFocus` draw the focus rectangles in the appropriate state. The call to `doDrawSelections` simply draws the current list selections in the rectangle at the bottom of the window.

The calls to TextMode if the program is running on Mac OS X are required because of

certain machinations in the function doDrawFrameAndFocus.

doActivateDeactivate

`doActivateDeactivate` is called when the `kEventWindowActivated` and `kEventWindowDeactivated` event types are received.

If the program is running on Mac OS 8/9 the root control is activated or deactivated, as appropriate, thus activating or deactivating all the controls in the window. (This is done automatically on Mac OS X.)

If the window is becoming active, the following occurs. For both lists, `LActivate` is called with true passed in the first parameter so as to highlight the currently selected cells. The calls to TextMode and `LUpdate` when the program is running on Mac OS 8/9 are required for cosmetic purposes only. `LUpdate` causes a redraw of all of the list's text in the `srcOr` mode. The calls to `doDrawFrameAndFocus` draw the list box frames in the active state and ensure that a keyboard focus frame is redrawn around the currently active list. The call to `doResetTypeSelection` resets certain variables used by the type selection function. (This latter is necessary because it is possible that, while the application was in the background, the user may have changed the Delay Until Repeat setting in the Keyboard control panel (Mac OS 8/9) or System Preferences/Keyboard (Mac OS X), a value which is used in the type selection function.) doDrawSelections redraws the current list selections in the `srcOr` mode.

Except for the call to `doResetTypeSelection`, much the same occurs if the window is becoming inactive, except that `LActivate` removes highlighting from the currently selected cells, `LUpdate` redraws the lists' text in the `grayishTextOr` mode (on Mac OS 8/9), `doDrawFrameAndFocus` removes the keyboard focus frame from the active list and draws the list box frames in the inactive state, and `doDrawSelections` redraws the current list selections in the `grayishTextOr` mode.

doInContent

`doInContent` is called when the `kEventWindowClickContentRgn` event type is received. Note that the mouse location received in the `mouseLocation` formal parameter is in local coordinates.

The calls to `GetListViewBounds` get the lists' display rectangles.

If the mouse click was in one of the list rectangles, and if that rectangle is not the current list's rectangle, `doRotateCurrentList` is called to change the currently active list. Next, `LClick` is called to handle all user action until the mouse-button is released. If `LClick` returns true, a double-click occurred, in which case `doExtractSelections` is called to extract and display the contents of the currently selected cells.

doControlHit

`doControlHit` is called when the `kEventMouseDown` event type is received and a call to `FindControlUnderMouse` reports that there is a control under the mouse cursor.

If the control is the Extract push button, `TrackControl` is called to handle user actions until the mouse button is released. If the cursor is still within the control when the mouse button is released, `doExtractSelections` is called to extract and display the contents of the currently selected cells.

If the control is one of the lists' scroll bars, `doRotateCurrentList` is called to change the currently active list. (This is necessary because the function `doInContent` only responds to clicks in the lists' display rectangles, which exclude the scroll bars.) `LClick` is then called to handle user interaction with the scroll bar.

doCreateTextList

`doCreateTextList`, supported by the two following functions, creates the text list.

`SetRect` sets the rectangle which will be passed as the `rDataBnds` parameter of the `LNew` call to specify one column and (initially) no rows. `SetPt` sets the variable that will be passed as the `cellSize` parameter so as to specify that the List Manager should automatically calculate the cell size. The next line adjusts the received list rectangle to eliminate the area occupied by the vertical scroll bar.

The call to `LNew` creates the list. The parameters specify that the List Manager is to calculate the cell size, the default list definition function is to be used, automatic drawing mode is to be enabled, no room is to be left for a size box, the list is not to have a horizontal scroll bar, and the list is to have a vertical scroll bar.

The call to `doAddRowsAndDataToTextList` adds rows to the list and stores data in its cells.

The next two lines set the cell at the topmost row as the initially selected cell. `doResetTypeSelection` calls a function which initializes certain variables used by the type selection function. The last line returns the handle to the list.

doAddRowsAndDataToTextList

`doAddRowsAndDataToTextList` adds rows to the text list and stores data in its cells. The data is retrieved from a `'STR#'` resource.

The for loop copies the strings from the specified `'STR#'` resource and passes each string

as a parameter in a call to `doAddTextItemsAlphabetically`, which inserts a new row into the list and copies the string to that cell.

Note at this point that the strings in the `'STR#'` resource are not arranged alphabetically.

doAddTextItemAlphabetically

`doAddTextItemAlphabetically` does the heavy work in the process of adding the rows to the text list and storing the text. The bulk of the code is concerned with building the list in such a way that the cells are arranged in alphabetical order.

The first line sets the variable found to `false`. The next line sets the variable `totalRows` to the number of rows in the list. (In this program, this is initially 0.) The next line sets the variable `currentRow` to -1.

The while loop executes until the variable found is set to `true`.

Within the loop, the first line increments `currentRow` to 0. The first time this function is called, `currentRow` will equal `totalRows` at this point and the loop will thus immediately exit to the first line below the loop. The call to `LAddRow` at this line adds one row to the list, inserting it before the row specified by `currentRow`. The list now has one row (cell (0,0)). `LSetCell` copies the string to this cell. The function then exits, to be re-called by `doAddRowsAndDataToTextList` for as many times as there are remaining strings.

The second time the function is called, the first line in the while loop again sets `currentRow` to 0. This time, however, the if block does not execute because `totalRows` is now 1. Thus `SetPt` sets the variable `aCell` to (0,0) and `LGetCellDataLocation` retrieves the offset and length of the data in cell (0,0). This allows the string in this cell to be alphabetically compared with the "incoming" string using `CompareText`. If the incoming string is "less than" the string in cell (0,0), `CompareText` returns -1, in which case

- The loop exits. `LaddRow` inserts one row before cell(0,0) and the old cell (0,0) thus becomes cell(0,1). The list now contains two rows.
- `SetPt` sets cell (0,0) and `LSetCell` copies the "incoming" string to that cell. The "incoming" string, which was alphabetically "less than" the first string, is thus assigned to the correct cell in the alphabetical sense.
- The function then exits, to be called again for as many times as there are remaining strings.

If, on the other hand, `CompareText` returns 0 (strings equal) or 1 ("incoming" string "greater than" the string in cell (0,0)), the loop repeats. At the first line in the loop, `currentRow` is incremented to 1, which is equal to `totalRows`. Accordingly, the loop

exits immediately, `LAddRow` inserts a row before cell (0,1) (that is, cell (0,1) is created), `LSetCell` copies the "incoming" string to that cell, and the function exits, to be called again for as many times as there are remaining strings.

The ultimate result of all this is an alphabetically ordered list.

doCreateIconList

`doCreateIconList`, supported by the following function, creates the icon list. This list uses a custom list definition function; accordingly, `CreateCustomList`, rather than `LNew`, is used to create the list.

`SetRect` sets the rectangle that will be passed as the `dataBounds` parameter of the `CreateCustomList` call to specify one column and (initially) no rows. `SetPt` sets the variable that will be passed as the cellSize parameter so as to specify that the List Manager should make the cell size of all cells 52 by 52 pixels. The next line adjusts the list rectangle to reflect the area occupied by the vertical scroll bar. The line after that assigns the universal procedure pointer to the custom list definition function to the `userProc` field of the variable of type `ListDefSpec` that will be passed in the `theSpec` parameter of the `CreateCustomList` call.

The call to `CreateCustomList` creates the list. The parameters specify that the List Manager is to make all cell sizes 52 by 52 pixels, a custom list definition function is to be used, automatic drawing mode is to be enabled, no room is to be left for a size box, the list is not to have a horizontal scroll bar, and the list is to have a vertical scroll bar.

The next line assigns `lOnlyOne` to the `selFlags` field of the list structure, meaning that the List Manager's cell selection algorithm is modified so as to allow only one cell to be selected at any one time.

The call to `doAddRowsAndDataToIconList` adds rows to the list and stores data in its cells.

The next two lines select the cell at the topmost row as the initially selected cell. The last line returns the handle to the list.

doAddRowsAndDataToIconList

`doAddRowsAndDataToIconList` adds ten rows to the icon list and stores a handle to an icon suite in each of the eight cells.

The first two lines set the variable `rowNumber` to the current number of rows, which is 0.

The for loop executes ten times. Each time through the loop, the following occurs:

- `GetIconSuite` creates a new icon suite and fills it with icons with the specified resource ID and of the types specified in the last parameter (that is, large icons only).
- `LAddRow` inserts a new row in the list at the location specified by the variable `rowNumber`. `SetPt` sets this cell and `LSetCell` stores the handle to the icon suite as the cell's data. The last line increments the variable `rowNumber`, which is passed in the SetPt call.

doHandleArrowKey

`doHandleArrowKey` further processes Down Arrow and Up Arrow key presses. This is the first of eleven functions dedicated to the handling of key-down events.

Recall that `doHandleArrowKey`'s third parameter (`allowExtendSelect`) is set to true by the calling function (`doKeyDown`) only if the text list is the currently active list.

The first line sets the variable `moveToTopBottom` to false, which can be regarded as the default. At the next two lines, if the Command key was also down at the time of the Arrow key press, this variable is set to true.

If the text list is the currently active list and the Shift key was down, `doArrowKeyExtendSelection` is called; otherwise, `doArrowKeyMoveSelection` is called.

doArrowKeyMoveSelection

`doArrowKeyMoveSelection` further processes those Arrow key presses which occurred when either list was the currently active list but the Shift key was not down. The effect of this function is to deselect all currently selected cells and to select the appropriate cell according to, firstly, which Arrow key was pressed (Up or Down) and, secondly, whether the Command key was down at the same time.

The if statement calls `doFindSelectedCell`, which searches for the first selected cell in the specified list. That function returns `true` if a selected cell is found, or `false` if the list contains no selected cells.

If `true` is returned by that call, the variable currentSelection will hold the first selected cell. However, this could be changed by the second line within the if block if the key pressed was the Down-Arrow. `doFindLastSelectedCell` finds the last selected cell (which could, of course, be the same cell as the first selected cell if only one cell is currently selected). Either way, the variable `currentSelection` will now hold either the only cell currently selected, the first cell selected (if more than one cell is currently selected and the key pressed was the Up Arrow), or the last cell selected (if more than one

cell is currently selected and the key pressed was the Down Arrow).

With that established, `doFindNewCellLoc` determines the next cell to select, which will depend whether the Command key was down at the time of the key press (that is, on whether the `moveToTopBottom` parameter is `true` or `false`). The variable `newSelection` will contain the results of that determination.

`doSelectOneCell` then deselects all currently selected cells and selects the cell specified by the variable `newSelection`.

It is possible that the newly selected cell will be outside the list's display rectangle. Accordingly, `doMakeCellVisible`, if necessary, scrolls the list until the newly selected cell appears at the top or the bottom of the display rectangle.

doArrowKeyExtendSelection

`doArrowKeyExtendSelection` is similar to the previous function except that it adds additional cells to the currently selected cells. This function is called only when the text list is the currently active list and the Shift key was down at the time of the Arrow key press.

By the fifth line, the variable `currentSelection` will hold either the only cell currently selected, the first cell selected (if more than one cell is currently selected and the key pressed was the Up Arrow), or the last cell selected (if more than one cell is currently selected and the key pressed was the Down Arrow).

`doFindNewCellLoc` determines the next cell to select, which will depend on whether the Command key was down at the time of the key press (that is, on whether the `moveToTopBottom` parameter is `true` or `false`). The variable `newSelection` will contain the results of that determination. The similarities between this function and `doArrowKeyMoveSelection` end there.

At the next line, `LGetSelect` is called to check whether the cell specified by the variable `newSelection` is selected. If it is not, `LSetSelect` selects it. (This check by `LGetSelect` is advisable because, for example, the first-selected cell as this function is entered might be cell (0,0)-that is, the very top row. If the Up-Arrow was pressed in this circumstance `doFindNewCellLoc` returns cell (0,0) in the `newSelection` variable. There is no point in selecting a cell that is already selected.)

It is possible that the newly selected cell will be outside the list's display rectangle. Accordingly, `doMakeCellVisible`, if necessary, scrolls the list until the newly selected cell appears at the top or the bottom of the display rectangle.

doTypeSelectSearch

doTypeSelectSearch is the main type selection function. It is called from doKeyDown whenever the key pressed is not the Tab key, the Up Arrow key or the Down Arrow key.

The global variables gTSStruct, gTSResetThreshold, and gTSLastListHit are central to the operation of doTypeSelectSearch. gTSStruct holds the current type selection search string entered by the user and the time in ticks of the last key press. gTSResetThreshold holds the number of ticks which must elapse before type selection resets, and is dependent on the value the user sets in the Delay Until Repeat setting in the Keyboard control panel (Mac OS 8/9) or System Preferences/Keyboard (Mac OS X). gTSLastListHit holds a handle to the last list that type selection affected.

The first block will cause doResetTypeSelection, which resets type selection, to be called if either of the following situations prevail: if the list that is the target of the current key press is not the same as the list that was the target of the previous key press; if a number of ticks since the last key press is greater than the number stored in gTSResetThreshold; if the current length of the type selection string is 63 characters.

The next line stores the handle to the list which is the target of the current key press in gTSLastListHit so as to facilitate the comparison at the first if block the next time the function is called. The next line stores the time of the current key press in gTSLastKeyTime for the same purpose.

The call to TypeSelectNewKey extracts the character code from the message field of the event structure and adds the character to the tsrKeyStrokes field of gTSStruct. That field now holds all the characters received since the last type selection reset.

SetPt sets the variable theCell to represent the first cell in the list. This is passed as a parameter in the LSearch call, and specifies the first cell to examine. LSearch examines this cell and all subsequent cells in an attempt to find a match to the type selection string. If a match exists, the cell in which the first match is found will be returned in theCell parameter, LSearch will return true and the following three lines will execute.

Of those three lines, ordinarily only the call to LSetSelect (which deselects all currently selected cells and selects the specified cell) and the last line (which, if necessary, scrolls the list so that the newly selected cell is visible in the display rectangle) would be necessary. However, because the function doSelectOneCell has no effect unless there is currently at least one selected cell in the list, the call to doSelectOneCell is included to account for the situation where the user may have deselected all of the text list cells using Command-clicking or dragging.

The actual matching task is performed by the match (callback) function the universal procedure pointer to which is passed in the third parameter to the LSearch call. Note that the default match function has been replaced by the custom callback function

doSearchPartialMatch.

doSearchPartialMatch

doSearchPartialMatch is the custom match function called by LSearch, in the previous function, to attempt to find a match to the current type selection string. For the default function to return a match, the type selection string would have to match an entire cell's text. doSearchPartialMatch, however, only compares the characters of the type selection string with the same number of characters in the cell's text. For example, if the type selection string is currently "fr" and a cell with the text "Fractal Painter" exists, doSearchPartialMatch will report a match.

A comparison by IdenticalText (which returns 0 if the strings being compared are equal) is only made if the cell contains data and the length of that data is greater than or equal to the current length of the type selection string. If these conditions do not prevail, doSearchPartialMatch returns 1 (no match found). If these conditions do prevail, IdenticalText is called with, importantly, both the third and fourth parameters set to the current length of the type selection string. IdenticalText will return 0 if the strings match or 1 if they do not match.

doFindFirstSelectedCell

doFindFirstSelectedCell and the following four functions are general utility functions called by the previous Arrow key handling and type selection functions. doFindFirstSelectedCell searches for the first selected cell in a list, returning true if a selected cell is found and providing the cell's coordinates to the calling function.

SetPt sets the starting cell for the LGetSelect call. Because the first parameter in the LGetSelect call is set to true, LGetSelect will continue to search the list until a selected cell is found or until all cells have been examined.

doFindFirstSelectedCell returns true when and if a selected cell is found.

doFindLastSelectedCell

doFindLastSelectedCell finds the last selected cell in a list (which could, of course, also be the first selected cell if only one cell is selected).

If the call to doFindFirstSelectedCell reveals that no cells are currently selected, doFindlastSelectedCell simply returns. If, however, doFindFirstSelectedCell finds a selected cell, that cell is passed as the starting cell in the LGetSelect call.

As an example of how the rest of this function works, assume that the first selected cell is

(0,1), and that cell (0,4) is the only other selected cell. `LGetSelect` examines this cell and returns true, causing the loop to execute. The first line in the while loop thus assigns (0,1) to `theCell` and the next line increments aCell to (0,2). `LGetSelect` starts another search using (0,2) as the starting cell. Because cells (0,2) and (0,3) are not selected, `LGetSelect` advances to cell (0,4) before it returns. Because it has found another selected cell, `LGetSelect` again returns true, so the loop executes again. `aCell` now contains (0,4), and the first line in the while loop assigns that to `theCell`. Once again, `LNextCell` increments `aCell`, this time to (0,5).

This time, however, `LGetSelect` will return `false` because neither cell (0,5) nor any cell below it is selected. The loop thus terminates, `theCell` containing (0,4), which is the last selected cell.

doFindNewCellLoc

`doFindNewCellLoc` finds the new cell to be selected in response to Arrow key presses. That cell will be either one up or one down from the cell specified in the `oldCellLoc` parameter (if the Command key was not down at the time of the Arrow key press) or the top or bottom cell (if the Command key was down).

The first line gets the number of rows in the list. (Recall that the List Manager sets the `dataBounds.bottom` coordinate to one more than the vertical coordinate of the last cell.)

If the Command key was down (`moveToTopBottom` is `true`) and the key pressed was the Up Arrow, the new cell to be selected is the top cell in the list. If the key pressed was the Down Arrow key, the new cell to be selected is the bottom cell in the list.

If the Command key was not down and the key pressed was the Up Arrow key, and if the first selected cell is the top cell in the list, the new cell to be selected remains as set at the second line in the function; otherwise, the new cell to be selected is set as the cell above the first selected cell. If the key pressed was the Down Arrow key, and if the last selected cell is the bottom cell in the list, the new cell to be selected remains as set at the second line in the function; otherwise, the new cell to be selected is set as the cell below the last selected cell.

doSelectOneCell

`doSelectOneCell` deselects all cells in the specified list and selects the specified cell.

If no cells in the list are selected, the function returns immediately. Otherwise, the first selected cell is passed as the starting cell in the call to `LGetSelect`.

The while loop will continue to execute while a selected cell exists between the starting

cell specified in the `LGetSelect` call and the end of the list. Within the loop, if the current `LGetSelect` starting cell is not the cell specified for selection, that cell is deselected. When the loop exits, `LSetSelect` selects the cell specified for selection.

Note that defeating the deselection of the cell specified for selection if it is already selected (the if statement within the while loop) prevents the unsightly flickering which would occur as a result of that cell being deselected inside the loop and then selected again after the loop exits.

doMakeCellVisible

`doMakeCellVisible` checks whether a specified cell is within the list's display rectangle and, if not, scrolls the list until that cell is visible.

The first line gets a copy of the rectangle that encompasses the currently visible cells. (Note that this rectangle is in cell coordinates.) The if statement tests whether the specified cell is within this rectangle. If it is not, the list is scrolled as follows:

- If the specified cell is "below" the bottom of the display rectangle, the variable `dRows` is set to the difference between the cell's v coordinate and the value in the bottom field of the display rectangle, plus 1. (Recall that the List Manager sets the bottom field to one greater than the v coordinate of the last visible cell.)
- If the specified cell is "above" the top of the display rectangle, the variable `dRows` is set to the difference between the cell's v coordinate and the value in the top field of the display rectangle.

With the number of cells to scroll, and the direction to scroll, established, `LScroll` is called to effect the scroll.

doResetTypeSelection

`doResetTypeSelection` resets the global variables which are central to the operation of the type selection function `doTypeSelectSearch`.

The first line resets the `tsrKeyStrokes` and `tsrLastKeyTime` fields of `gTSStruct` to `NULL` and 0 respectively. The next line sets the variable which holds the handle to the list which is the target of the current key press to `NULL`. The next line sets the variable which holds the type selection reset threshold to twice the value stored in the low memory global variable `KeyThresh`. However, if this value is greater than the value represented by the constant `kMaxKeyThresh`, the variable is made equal to `kMaxKeyThresh`.

doRotateCurrentList

`doRotateCurrentList` rotates the currently active list in response to the Tab key and to mouse-downs in the non-active list.

The first line saves the handle to the currently active list. The next line retrieves the handle to the new list to be activated from the `refCon` field of the currently active list's list structure. The third line makes the new list the currently active list.

The last two lines cause the keyboard focus frame to be erased from the previously current list, the list box frame to be drawn around the previously current list, and the keyboard focus frame to be drawn around the new current list.

doDrawFrameAndFocus

`doDrawFrameAndFocus` is called by `doDrawContent`, `doActivateDeactivate`, and `doRotateCurrentList` to draw or erase the keyboard focus frame from the specified list, and to draw the list box frame in either the activated or deactivated state.

The second and third lines get the list's rectangle from the `rView` field of the list structure and expand it to the right by the width of the scroll bar.

The machinations at the next four lines are for cosmetic purposes only. If the program is running on Mac OS 8/9, the current background color and pattern will be white, so the saved background color/pattern must be restored before the first call to `DrawThemeFocusRect`, which erases the keyboard focus frame to the background color/pattern.

Depending on the value received in the `inState` formal parameter, the list box frame is drawn in either the activated or deactivated state. If the specified list is the current list, `DrawThemeFocusRect` is called again, this time to draw the keyboard focus frame.

If the program is running on Mac OS 8/9, the last two lines reset the backgound color/pattern to white.

doExtractSelections

`doExtractSelections` is called when the user clicks the Extract push button or double clicks an item in a list.

The first block gets the handles to the lists. The next two lines initialize the `Str255` array that will be used to hold the extracted strings.

The next block copies the data from the selected cells in the text list to the `Str255` array. The for loop is traversed once for each cell in the list. `SetPt` increments the v coordinate

of the variable `theCell`. If the specified cell is selected (`LGetSelect`), `LGetCellDataLocation` is called to get the length of the data in the cell, and `LGetCell` is called to copy the cell's data into an element of the `Str255` array.

The next block gets the selected cell in the icon list, retrieves the related string from the specified `STR#` resource, and assigns it to the 15th element of the `Str255` array. `SetPt` sets the starting cell for the `LGetSelect` search.

The last two lines force a `kEventWindowDrawContent` event, which will cause the function `doDrawSelections` to draw the contents of the `Str255` array in the group box at the bottom of the window.

doDrawSelections

`doDrawSelections` is called by `doDrawContent` and `doActivateDeactivate` to draw the contents of the `Str255` array "filled in" by the function `doExtractSelections`.

listDefFunction

`listDefFunction` the custom list definition (callback) function used by the window's icon list.

The List Manager sends a list definition function four types of messages in the message parameter. Only two of these are relevant to this list definition function. `listDefFunction` calls the appropriate function to handle each message type.

doLDEFDraw

`doLDEFDraw` handles the `lDrawMsg` message, which relates to a specific cell.

The first two lines save the current graphics port and set the graphics port to the port in which the list is drawn.

`EraseRect` erases the cell rectangle. The next line gets a copy of the 52-by-52 pixel cell rectangle. The next four lines adjust this rectangle to the size of a 32-by-32 pixel icon.

The if statement checks whether the cell's data is 4 bytes long (the size of a handle). If it is, `LGetCell` is called to get the cells's data into the variable `iconSuiteHdl` and *PlotIconSuite* is called to draw the icon within the specified rectangle. If the list is active, `kTransformNone` is passed in the transform parameter, otherwise `kTransformDisabled` is passed. This latter causes the icon to be drawn in the disabled (dimmed) state.

GetIndString is then called to get the string corresponding to the icon. The rectangle used to draw the icon is adjusted and passed, together with the string, in a call to TETextBox. TETextBox draws the string, with center justification, underneath the icon.

If the lDrawMsg message indicated that the cell was selected, the cell highlighting function is called. The previously saved graphics port is then restored

doLDEFHighlight

doLDEFHighlight handles the lHiliteMsg message and may also be called from doLDEFDraw.

A copy of the value in the low memory global HiliteMode is acquired, BitClr is called to clear the highlight bit, and HiliteMode is set to this new value. The last line highlights the cell.

DialogList.c

doListsDialog contains the main functions pertaining to the lists in the movable modal dialog.

doListsDialog

doListsDialog creates a movable modal dialog using 'DLOG', 'dlgx', 'dftb', and 'DITL' resources. The 'DITL' resource contains two list controls. Each list control is supported by an 'ldes' resource. Both 'ldes' resources specify no rows, one column, a cell height of 14 pixels, a vertical scroll bar, and the system LDEF. The 'dftb' resource specifies the small system font for the list controls.

At the first block, the window, if open, is explicitly deactivated. The dialog is then created. At the next block, the Dialog Manager is told which items are the default and Cancel items.

A custom event filter function is used. The call to NewModalFilterProc creates the associated universal procedure pointer.

At the next block, and for each list control, the handle to the list control is obtained, the handle to the associated list structure is obtained, the function doAddRowsAndDataToTextList is called to add the specified number of rows and the data to the list's cells, the cell-selection algorithm is customized to allow the selection of one cell only, and the first cell is selected.

ShowWindow is then called to display the dialog. The call to SetKeyboardFocus sets the keyboard focus to the "Date Format" list.

Within the do-while loop, `ModalDialog` retains control until an enabled item is hit. If the push buttons are hit, or if the last click in one of the list boxes was a double-click, the loop exits.

If the item hit is the "Date Format" list, `SetPt` sets the variable `theCell` to represent the first cell in the list. This is passed as a parameter in the `LGetSelect` call, which searches the list until it finds a cell that is selected. `LGetDataLocation` is called to get the length of the data in that cell and `LGetCell` is called to copy the data (a string) to a local `Str255` variable. At the next block, a reference to the static text control associated with this list is obtained and its text is set with the string obtained by `LGetCell`. `Draw1Control` is then called to draw the static text field control with this newly set text.

The last action is to check whether the last click in the list box was a double-click. If the last click was a double-click, the variable `wasDoubleClick` is set to true, causing the loop to exit.

The same general procedure is followed in the event of a hit on the "Watermark" list.

When the OK push button is hit, or one of the lists has been double-clicked, the dialog and the universal procedure pointer are disposed of.

eventFilter

A custom event filter function is used to intercept `keyDown` events so as to support type selection in the "Watermark" list.

If the event is a keyDown event, the character code is extracted from the event structure's message field.

If the key hit was not the Up-Arrow, Down-Arrow, or tab key, the following occurs. `GetDialogItemAsControl` is called to get the reference to the "Watermark" list control, `GetControlData` is called to get the handle to the associated list, and `GetKeyboardFocus` is called to get the reference to the control with keyboard focus. If the "Watermark" list control currently has the focus, the function `doTypeSelectSearch` is called (to handle type selection) and `Draw1Control` is called on the list control to ensure that the type-selected item is highlighted. `handledEvent` is then set to true to inform `ModalDialog` that the filter function handled the event.

Apart from supporting type-selection in the "Watermark" list, this arrangement means that the only `keyDown` events received by `ModalDialog` in respect of the "Date Format" list will be Up-Arrow, Down-Arrow, and tab key events.

Chapter 23. Drag and Drop

IN THIS CHAPTER

- Overview
- Drag Items, Drag Item Flavors, and Handlers
- The Drag Region and The Drag Image
- Drag Functions-Overriding Drag Manager Default Behavior
- Main Drag Manager Constants, Data Types, and Functions
- Relevant TextEdit Function
- Demonstration Program Drag Listing
- Demonstration Program Drag Comments

Overview

The Drag Manager

Using the Drag Manager, you can provide your application with the capability to drag objects from your application's windows, to your application's windows, and within your application's windows.

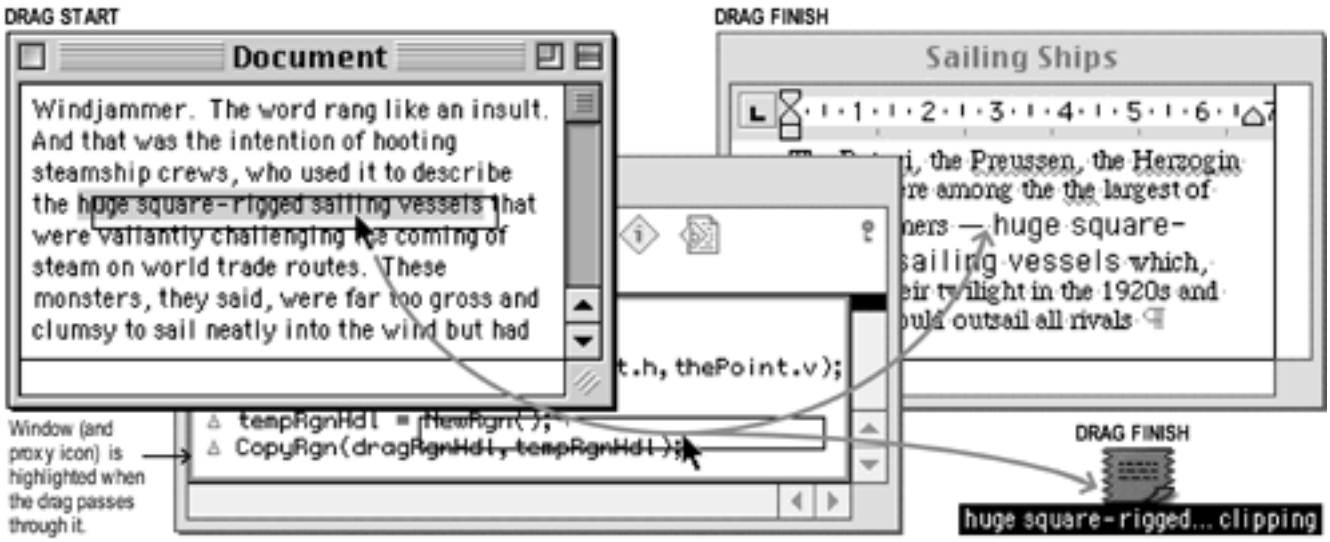
The Three Phases of a Drag

All drag-and-drop operations comprise the following three distinct phases:

- Starting the drag.
- Tracking the drag.
- Finishing the drag.

Different applications may be involved in each of these three phases. For example, when the user drags an item from one application to another, the source application starts the drag, other applications through whose windows the user drags the item may track the drag, and the application owning the target window finishes the drag by accepting the drop (see Fig 1).

Figure 1. STARTING, TRACKING, AND FINISHING A DRAG IN DIFFERENT APPLICATIONS


DRAG START
Document
Windjammer. The word rang like an insult. And that was the intention of hooting steamship crews, who used it to describe the huge square-rigged sailing vessels that were valiantly challenging the coming of steam on world trade routes. These monsters, they said, were far too gross and clumsy to sail neatly into the wind but had
DRAG FINISH
Sailing Ships
Window (and proxy icon) is highlighted when the drag passes through it.
DRAG FINISH
huge square-rigged... clipping

Of course, when the drag and drop is wholly within one of an application's document windows, that single application starts the drag, tracks it through the window, and accepts the drop.

Starting the Drag

A drag is initiated when the user clicks on a selected object, keeps the mouse button down, and begins to move the mouse (see the left hand window in Fig 1). Using the function `WaitMouseMoved`, you can determine if the mouse has moved far enough to initiate a drag. If `WaitMouseMoved` returns `true`, your application should call `NewDrag` to create a new **drag reference.** A drag reference is an opaque data type that you use to refer to a specific drag process.

Having created a drag reference, and for each item in the drag, you must then add at least one **drag item flavor** to that reference using the function `AddDragItemFlavor`. Drag item flavors represent the various data formats in which each individual item may be produced. The concept is similar to that applying to Scrap flavors as described in Chapter 20.

With the drag item flavor (or flavors) added, your application can then initiate tracking by calling the function `TrackDrag`.

Tracking the Drag-Drag Tracking Handlers

The Drag Manager tracks the drag through each window the cursor moves over as the user drags the selection across the screen. During this process, the user is presented with the following feedback:

- Destination highlighting will occur when the cursor has left the source location. (See the middle window in Fig 1, whose owner application supports drag and drop and can accept the drop.)
- If a container under the cursor (such as a folder in the Finder) can accept the drop, that container will be highlighted.

As the cursor moves through the middle window in Fig 1, for example, the Drag Manager calls a function, called a **drag tracking handler,** defined by that window's owner application. Drag tracking handlers inspect the description of the item, or items, being dragged and highlight the window if this inspection reveals that the window can accept the drop. The application's drag tracking handler uses the functions `ShowDragHilite` and `HideDragHilite` to create and remove the highlighting. (The acceptable drop region, and thus the region highlighted, does not necessarily have to be the whole of the window's content region.)

Finishing the Drag-Drag Receive Handlers

When the user releases the mouse button in the right hand window in Fig 1, for example, the Drag Manager calls a function, called a **drag receive handler,** defined by that window's owner

application. Drag receive handlers accept the drop and insert the selection at its final destination.

Drag Items, Drag Item Flavors, and Handlers

Drag Items and Item Reference Numbers

A **drag item** is a single distinct object, which could be, for example, a selection from a painting in a painting program, a selection of objects in a drawing program, or a continuous range of text in a text editing program. A discontinuous selection of text (resulting from using the Command key in some programs) would result in multiple drag items.

A new drag item is created if the **item reference number** passed in the `theItemRef` parameter of a call to `AddDragItemFlavor` is different from any other item reference number.

> **Note**
>
> In many cases it is easiest to use index numbers as item reference numbers (for example, 1, 2, 3…). Item reference numbers are only used as unique "key" numbers for each item. Item reference numbers do not need to be given in order, nor must they be sequential. Depending on your application, it might be easier to use your own internal memory addresses as item reference numbers (as long as each item being dragged has a unique item reference number).

Drag Item Flavors

Any item that can be dragged can generally be represented using several different flavors (that is, data formats). At the start of a drag, your application must inform the Drag Manager, using the function `AddDragItemFlavor`, of the flavors that you can provide to the receiver of the drag. For example, if the drag item is a selection of text, you might elect to provide the data in both the standard `'TEXT'` format and in the Rich Text Format (RTF). Alternatively, instead of supplying the data in RTF, you might supplement the `'TEXT'` data with `'styl'` data. You might also provide the data in your application's own internal data format to cater for a drag and drop to one of your application's documents, including the source document.

Your application adds additional flavors to an item by passing the same item reference number in the `AddDragItemFlavor` call as was used in the `AddDragItemFlavor` call which created the item and added the first flavor.

Different destinations may prefer different data formats, and there is no certainty as to whether a drag will contain the preferred, or an acceptable, flavor. Thus, as previously stated, the

destination application needs to inspect the description of the items being dragged to determine whether an acceptable flavor is available and only highlight the relevant window if it can accept a drop. The function `GetFlavorFlags` is used to determine whether the drag contains a specific flavor type.

Flavor Flags

The following flavor flags, which are used by the Drag Manager and its clients to provide additional information about a specific drag item flavor, may be passed in the `theFlags` parameter of the function `AddDragItemFlavor` and retrieved by the function `GetFlavorFlags`:

Flag	*Description*
`flavorSenderOnly`	Set by the sender if the flavor should only be available to the sender.
`flavorSenderTranslated`	Set if the flavor data is translated by the sender. Useful if the receiver needs to ascertain whether the sender is performing its own translation to generate this data type.
`flavorNotSaved`	Set by the sender if the flavor data should not be stored by the receiver. Useful for marking flavor data that will become stale after the drag is completed.
`flavorSystemTranslated`	Set if the flavor data is translated by the Translation Manager. If this flavor is requested, the Drag Manager will acquire the required data type from the sender and then use the Translation Manager to provide the data requested by the receiver.

Note

Note that the Finder does not save flavor types marked with the `flavorSenderTranslated`, `flavorNotSaved`, and `flavorSystemTranslated` flags into clippings files.

Drag Handlers

Drag tracking handlers and drag receive handlers are callback functions that your application registers with the Drag Manager using the functions `InstallTrackingHandler` and `InstallReceiveHandler`. The window reference of the window on which the handlers are to be installed is passed in the `theWindow` parameter of the two handler installer functions. You can install more than one tracking handler and more than one receive handler on

the same window, in which case the Drag Manager calls each of the handlers in the order they were installed.

If you pass `NULL` in the `theWindow` parameter of the handler installer functions, the Drag Manager will register the handler in a special area that is used when the drag occurs in any window of your application. Handlers installed in this special area are called **default handlers.**

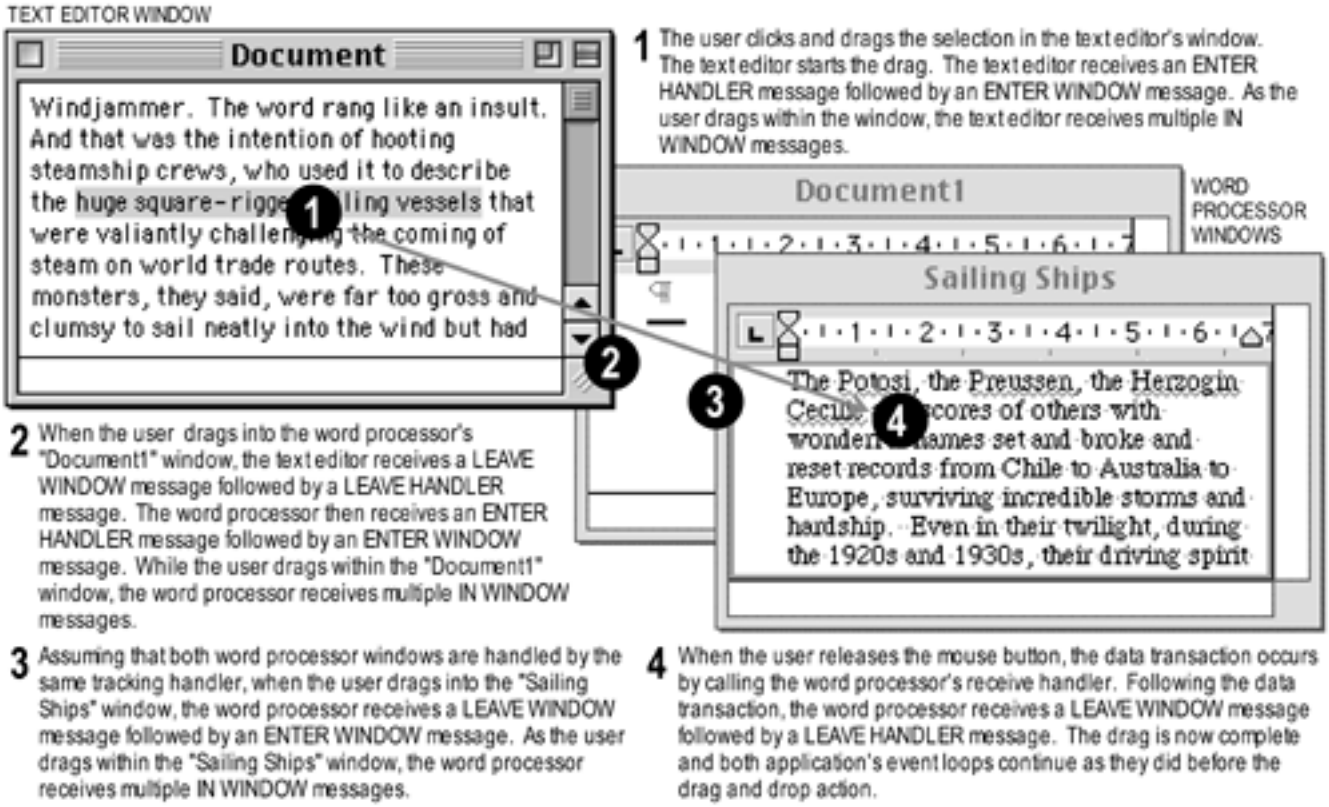
Drag Tracking Handlers

As the user moves the mouse through your application's windows during the drag, the Drag Manager sends status messages to your tracking handler. These messages are as follows:

Message	***Description***
Enter handler	Received when the focus of the drag enters a window handled by your tracking handler from a window not handled by your tracking handler.
Enter window	Received when the focus of a drag enters any window handled by your tracking handler.
In window	Received as the user drags within a window handled by your tracking handler.
Leave window	Received when the focus of a drag leaves a window that is handled by your tracking handler.
Leave handler	Received when the focus of a drag enters a window that is not handled by your tracking handler.

Fig 2 illustrates the receipt of tracking messages involving multiple applications and windows.

Figure 2. DRAG TRACKING THROUGH MULTIPLE APPLICATIONS AND WINDOWS


TEXT EDITOR WINDOW
Document
Windjammer. The word rang like an insult. And that was the intention of hooting steamship crews, who used it to describe the huge square-rigg ling vessels that were valiantly challeng the coming of steam on world trade routes. These monsters, they said, were far too gross and clumsy to sail neatly into the wind but had
1 The user clicks and drags the selection in the text editor's window. The text editor starts the drag. The text editor receives an ENTER HANDLER message followed by an ENTER WINDOW message. As the user drags within the window, the text editor receives multiple IN WINDOW messages.
Document1
WORD PROCESSOR WINDOWS
Sailing Ships
The Potosi, the Preussen, the Herzogin Cecil cores of others with wonder names set and broke and reset records from Chile to Australia to Europe, surviving incredible storms and hardship. Even in their twilight, during the 1920s and 1930s, their driving spirit
2 When the user drags into the word processor's "Document1" window, the text editor receives a LEAVE WINDOW message followed by a LEAVE HANDLER message. The word processor then receives an ENTER HANDLER message followed by an ENTER WINDOW message. While the user drags within the "Document1" window, the word processor receives multiple IN WINDOW messages.
3 Assuming that both word processor windows are handled by the same tracking handler, when the user drags into the "Sailing Ships" window, the word processor receives a LEAVE WINDOW message followed by an ENTER WINDOW message. As the user drags within the "Sailing Ships" window, the word processor receives multiple IN WINDOW messages.
4 When the user releases the mouse button, the data transaction occurs by calling the word processor's receive handler. Following the data transaction, the word processor receives a LEAVE WINDOW message followed by a LEAVE HANDLER message. The drag is now complete and both application's event loops continue as they did before the drag and drop action.

Drag Receive Handlers

When the user drops an item, or a collection of items, in one of your application's windows, the Drag Manager calls any receive handlers installed on the destination window. Your receive handler can then request the drag item flavor, or flavors, that you wish to accept and insert the data at its final destination.

When the drop occurs within the window in which the drag originated, and the user did not press the Option key either before the mouse button went down or before it was released, receive handlers must delete the original selection before inserting the dropped data. If the option button was not pressed, receive handlers must not delete the selection.

The Drag Region and The Drag Image

Creating the Drag Region

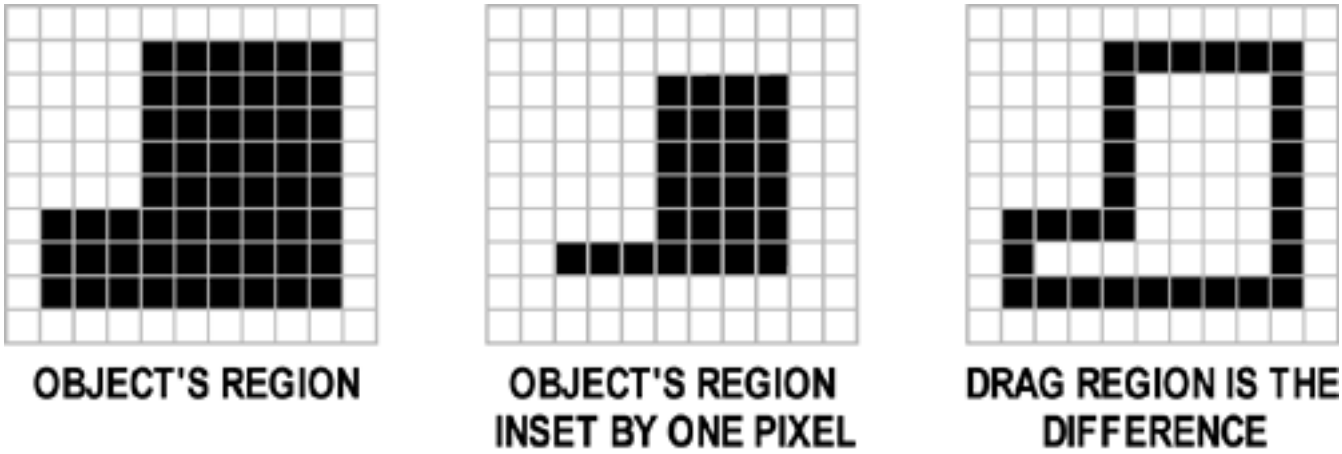
Recall that your application initiates tracking by calling the function `TrackDrag`. This function takes a region, specifically, the **drag region,** in its `theRgn` parameter. This is the region, drawn by the Drag Manager in the dithered gray pattern (Mac OS 8/9) or gray color (Mac OS X), that moves with the mouse during the drag (see Fig 1).

As an example, to create a drag region when a single item is selected, your application should

- Copy the drag item's region to a temporary region and inset the temporary region by 1 pixel.
- Use `DiffRgn` to subtract the temporary region from a copy of the item's region.

The resulting copy of the item's region has the same outline as the item's original region but is only one pixel thick (see Fig 3).

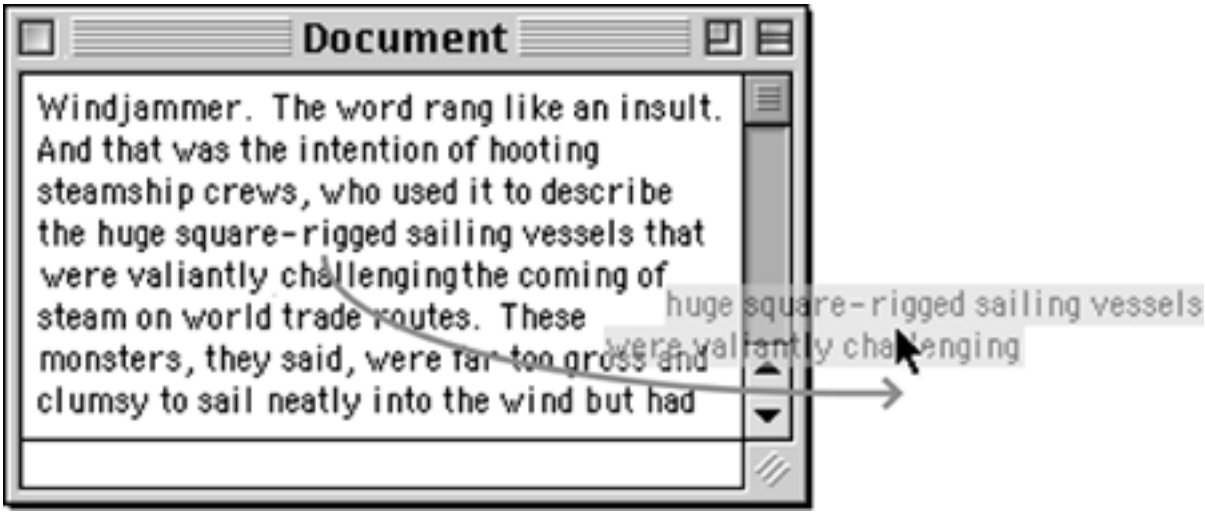
Figure 3. CREATING THE DRAG REGION



Setting the Drag Image-Translucent Drags

The drag region may also be visually represented to the user by a translucent image of the original selection (see Fig 4).

Figure 4. A TRANSLUCENT DRAG


Document
Windjammer. The word rang like an insult. And that was the intention of hooting steamship crews, who used it to describe the huge square-rigged sailing vessels that were valiantly challengingthe coming of steam on world trade routes. These monsters, they said, were far too gross and clumsy to sail neatly into the wind but had
huge square-rigged sailing vessels
were valiantly challenging

To support translucent dragging of a text selection on Mac OS 8/9, you should

- Create an offscreen graphics world the size of the bounding rectangle of the highlight region. The offscreen graphics world should be eight bits deep.
- Copy the area of the window enclosed by the highlight region's bounding rectangle to the offscreen graphics world.
- Create a region for a mask that is exactly the same shape as the highlight region. (Note that if you do not supply a mask, the entire rectangular area of the offscreen pixel map, including white space, will be dragged.)
- Call `SetDragImage` to associate the image with the drag reference, passing in the handle to the offscreen pixel map, the handle to the mask region, and a constant specifying the required level of translucency. `SetDragImage` installs a custom drawing function (see below) to perform the translucent drawing.

The level of translucency may be specified using one of the following drag image flags:

Constant	*Value*	*Description*
`kDragStandardTranslucency`	`0L`	65% image translucency. This is the recommended level.
`kDragDarkTranslucency`	`1L`	50% image translucency.
`kDragDarkerTranslucency`	`2L`	25% image translucency.
`kDragOpaqueTranslucency`	`3L`	0% image translucency (opaque).

Drag Functions-Overriding Drag Manager Default Behavior

If you wish to override the Drag Manager's default behavior, you can supply it with several different kinds of drag function. Only the sender can specify drag functions.

Drawing Function

When a drawing function is installed, the Drag Manager sends that function a sequence of messages that allow the application to assume responsibility for drawing the drag region (or similar feedback) on the screen. If translucent dragging is used, you must not install a custom drawing function.

Send Data Function

If you install a send data function, the Drag Manager calls that function when the receiver application requests a drag item flavor that the Drag Manager does not currently have the data cached for.

Ordinarily, the Drag Manager caches the flavor data for all flavors that were added to the drag with the `AddDragItemFlavor` function. If a receiver calls `GetFlavorData` to get a flavor's data, the Drag Manager simply returns the cached data to the receiver.

However, if your application passes `NULL` as the pointer to the flavor data in the `AddDragItemFlavor` call, the Drag Manager does not cache the data. In effect, your application has made a "promise" to later supply the data in the event that a receiver requests it. Thus, if a receiver calls `GetFlavorData` for that particular flavor, the Drag Manager will call your send data function to get the data from the sender. Your send data function should then create that flavor's data and call `SetDragItemFlavorData` to send the data to the Drag Manager.

This mechanism of "promising" data that may, in the event, not be called for by a receiver is useful where the sender must perform expensive computations to produce the data or if the resulting data requires a large amount of memory to store.

Drag Input Function

If you install a **drag input function,** the Drag Manager calls that function when sampling the mouse position and keyboard state to allow the application to override the current state of the input devices. The Drag Manager passes the current mouse location, mouse button state, and keyboard modifier status to your drag input function, which can then modify

those parameters.

Main Drag Manager Constants, Data Types, and Functions

Constants

Flavor Flags

```
flavorSenderOnly                = (1 << 0)
flavorSenderTranslated          = (1 << 1)
flavorNotSaved                  = (1 << 2)
flavorSystemTranslated          = (1 << 8)
```

Drag Attributes

```
kDragHasLeftSenderWindow        = (1L << 0)
kDragInsideSenderApplication    = (1L)
kDragInsideSenderWindow         = (1L << 2)
```

Drag Tracking Handler Messages

```
kDragTrackingEnterHandler       = 1
kDragTrackingEnterWindow        = 2
kDragTrackingInWindow           = 3
kDragTrackingLeaveWindow        = 4
kDragTrackingLeaveHandler       = 5
```

Drag Image Flags

```
kDragStandardTranslucency       = 0L
kDragDarkTranslucency           = 1L
kDragDarkerTranslucency         = 2L
kDragOpaqueTranslucency         = 3L
```

Drag Drawing Handler Messages

```
kDragRegionBegin                = 1
kDragRegionDraw                 = 2
kDragRegionHide                 = 3
kDragRegionIdle                 = 4
kDragRegionEnd                  = 5
```

Data Types

```
typedef struct OpaqueDragRef *DragRef;
typedef UInt32  DragItemRef;
typedef OSType  FlavorType;
typedef UInt32  DragAttributes;
typedef UInt32  FlavorFlags;
typedef SInt16  DragTrackingMessage;
typedef SInt16  DragRegionMessage;
typedef UInt32  DragImageFlags;
```

Functions

Installing and Removing Drag Handlers

```
OSErr   InstallTrackingHandler(DragTrackingHandlerUPP trackingHandler,WindowRef theWindow,
        void *handlerRefCon);
```

```
OSErr   InstallReceiveHandler(DragReceiveHandlerUPP receiveHandler,WindowRef theWindow,
        void *handlerRefCon);
OSErr   RemoveTrackingHandler(DragTrackingHandlerUPP trackingHandler,WindowRef theWindow);
OSErr   RemoveReceiveHandler(DragReceiveHandlerUPP receiveHandler,WindowRef theWindow);
```

Creating and Disposing of Drag References

```
OSErr   NewDrag(DragRef *theDrag);
OSErr   DisposeDrag(DragRef theDrag);
```

Adding Drag Item Flavors

```
OSErr   AddDragItemFlavor(DragRef theDrag,DragItemRef theItemRef,FlavorType theType,
        const void *dataPtr,Size dataSize,FlavorFlags theFlags);
OSErr   SetDragItemFlavorData(DragRef theDrag,DragItemRef theItemRef,FlavorType theType,
        const void *dataPtr,Sized ataSize,UInt32 dataOffset);
```

Performing a Drag

```
OSErr   TrackDrag(DragRef theDrag,const EventRecord *theEvent,RgnHandle theRegion);
```

Getting Drag Item Information

```
OSErr   CountDragItems(DragRef theDrag,UInt16 *numItems);
OSErr   GetDragItemReferenceNumber(DragRef theDrag,UInt16 index,DragItemRef *theItemRef);
OSErr   CountDragItemFlavors(DragRef theDrag,DragItemRef theItemRef,UInt16 *numFlavors);
OSErr   GetFlavorType(DragRef theDrag,DragItemRef theItemRef,UInt16 index,FlavorType
*theType);
OSErr   GetFlavorFlags(DragRef theDrag,DragItemRef theItemRef,FlavorType theType,
        FlavorFlags *theFlags);
OSErr   GetFlavorDataSize(DragRef theDrag,DragItemRef theItemRef,FlavorType theType,
        Size *dataSize);
OSErr   GetFlavorData(DragRef theDrag,DragItemRef theItemRef,FlavorType theType,void *
dataPtr,
        Size *dataSize,UInt32 dataOffset);
```

Getting and Setting Drag Status Information

```
OSErr   GetDragAttributes(DragRef theDrag,DragAttributes *flags);
OSErr   GetDragMouse(DragRef theDrag,Point *mouse,Point *globalPinnedMouse);
OSErr   SetDragMouse(DragRef theDrag,Point globalPinnedMouse);
OSErr   GetDragOrigin(DragRef theDrag,Point *globalInitialMouse);
OSErr   GetDragModifiers(DragRef theDrag,SInt16 *modifiers,SInt16 *mouseDownModifiers,
        SInt16 *mouseUpModifiers);
```

Window Highlighting Utilities

```
OSErr   ShowDragHilite(DragRef theDrag,RgnHandle hiliteFrame,Boolean inside);
OSErr   HideDragHilite(DragRef theDrag);
OSErr   UpdateDragHilite(DragRef theDrag,RgnHandle updateRgn);
```

Drag Manager Utilities

```
Boolean WaitMouseMoved(Point initialMouse);
```

Setting Drag Functions

```
OSErr   SetDragSendProc(DragRef theDrag,DragSendDataUPP sendProc,void *dragSendRefCon);
OSErr   SetDragInputProc(DragRef theDrag,DragInputUPP inputProc,void *dragInputRefCon);
OSErr   SetDragDrawingProc(DragRef theDrag,DragDrawingUPP drawingProc,void
*dragDrawingRefCon);
```

Setting the Drag Image-Translucent Dragging

```
OSErr   SetDragImage (DragRef theDrag, PixMapHandle imagePixMap,RgnHandle imageRgn,
                      Point imageOffsetPt,DragImageFlags theImageFlags);
```

Creating and Disposing of Universal Procedure Pointers

```
DragTrackingHandlerUPP  NewDragTrackingHandlerUPP(DragTrackingHandlerProcPtr userRoutine);
DragReceiveHandlerUPP   NewDragReceiveHandlerUPP(DragReceiveHandlerProcPtr userRoutine);
DragSendDataUPP         NewDragSendDataUPP(DragSendDataProcPtr userRoutine);
DragInputUPP            NewDragInputUPP(DragInputProcPtr userRoutine);
DragDrawingUPP          NewDragDrawingUPP(DragDrawingProcPtr userRoutine);
void    DisposeDragTrackingHandlerUPP(DragTrackingHandlerUPP userUPP);
void    DisposeDragReceiveHandlerUPP(DragReceiveHandlerUPP userUPP);
void    DisposeDragSendDataUPP(DragSendDataUPP userUPP);
void    DisposeDragInputUPP(DragInputUPP userUPP);
void    DisposeDragDrawingUPP(DragDrawingUPP userUPP);
```

Application Defined (Callback) Functions

```
OSErr   myDragTrackingHandler(DragTrackingMessage message,WindowRef theWindow,
        void *handlerRefCon,DragRef theDrag);
OSErr   myDragReceiveHandler(WindowRef theWindow,void *handlerRefCon,DragRef theDrag);
OSErr   muDragSendDataFunction(FlavorType theType,void *dragSendRefCon,DragItemRef
theItemRef,
        DragRef theDrag);
OSErr   myDragInputFunction(Point *mouse,SInt16 *modifiers,void *dragInputRefCon,
        DragRef theDrag);
OSErr   myDragDrawingFunction(DragRegionMessage message,RgnHandle showRegion,
        Point showOrigin,RgnHandle hideRegion,Point hideOrigin,void *dragDrawingRefCon,
        DragRef theDrag);
```

Relevant TextEdit Function

```
OSErr   TEGetHiliteRgn(RgnHandle region,TEHandle hTE);
```

Demonstration Program Drag Listing

```
// ****************************************************************************************
// Drag.h                                                                  CARBON EVENT MODEL
// ****************************************************************************************
//
// This program demonstrates drag and drop utilising the Drag Manager. Support for Undo and
// Redo of drag and drop within the source window is included.
//
// The bulk of the code in the source code file Drag.c is identical to the code in the file
// Text1.c in the demonstration program MonoTextEdit (Chapter 21).
//
// The program utilises the following resources:
//
//    A 'plst' resource.
//
// •  An 'MBAR' resource, and 'MENU' resources for Apple, File, and Edit menus.
//
// •  A 'WIND' resource (purgeable) (initially visible).
//
// •  'CNTL' resources (purgeable) for the vertical scroll bars in the text editor window and
//    Help dialog, and for the pop-up menu in the Help Dialog.
//
// •  A 'STR#' resource  (purgeable) containing error text strings.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// ****************************************************************************************

// ..................................................................... includes

#include <Carbon.h>

// ..................................................................... defines

#define rMenubar          128
#define mAppleApplication 128
#define  iAbout           1
#define mFile             129
#define  iNew             1
#define  iOpen            2
#define  iClose           4
#define  iSaveAs          6
```

```
#define  iQuit           12
#define mEdit            130
#define  iUndo           1
#define  iCut            3
#define  iCopy           4
#define  iPaste          5
#define  iClear          6
#define  iSelectAll      7
#define rWindow          128
#define rVScrollbar      128
#define rErrorStrings    128
#define  eMenuBar        1
#define  eWindow         2
#define  eDocStructure   3
#define  eTextEdit       4
#define  eExceedChara    5
#define  eNoSpaceCut     6
#define  eNoSpacePaste   7
#define  eDragHandler    8
#define  eDrag           9
#define  eDragUndo       10
#define kMaxTELength     32767
#define kTab             0x09
#define kBackSpace       0x08
#define kForwardDelete   0x7F
#define kReturn          0x0D
#define kEscape          0x1B
#define kReturn          0x0D
#define kFileCreator     'KJbb'
#define topLeft(r)       (((Point *) &(r))[0])
#define botRight(r)      (((Point *) &(r))[1])

// ..................................................................................... typedefs

typedef struct
{
  TEHandle   textEditStrucHdl;
  ControlRef vScrollbarRef;
  WindowRef  windowRef;
  Boolean    windowTouched;
  Handle     preDragText;
  SInt16     preDragSelStart;
  SInt16     preDragSelEnd;
  SInt16     postDropSelStart;
  SInt16     postDropSelEnd;
}  docStructure, *docStructurePointer;

// ................................................................... function prototypes
```

```
void            main                    (void);
void            doPreliminaries         (void);
OSStatus        appEventHandler         (EventHandlerCallRef,EventRef,void *);
OSStatus        windowEventHandler      (EventHandlerCallRef,EventRef,void *);
void            doIdle                  (void);
void            doKeyEvent              (SInt8);
void            scrollActionFunction    (ControlRef,SInt16);
void            doInContent             (Point,EventRecord *,Boolean);
void            doDrawContent           (WindowRef);
void            doActivateDeactivate    (WindowRef,Boolean);
void            doOSEvent               (EventRecord *);
WindowRef       doNewDocWindow          (void);
EventHandlerUPP doGetHandlerUPP         (void);
Boolean         customClickLoop         (void);
void            doSetScrollBarValue     (ControlRef,SInt16 *);
void            doAdjustMenus           (void);
void            doMenuChoice            (MenuID,MenuItemIndex);
void            doFileMenu              (MenuItemIndex);
void            doEditMenu              (MenuItemIndex);
SInt16          doGetSelectLength       (TEHandle);
void            doAdjustScrollbar       (WindowRef);
void            doAdjustCursor          (WindowRef);
void            doCloseWindow           (WindowRef);
void            doSaveAsFile            (TEHandle);
void            doOpenCommand           (void);
void            doOpenFile              (FSSpec);
void            doErrorAlert            (SInt16);
void            navEventFunction        (NavEventCallbackMessage,NavCBRecPtr,
                                         NavCallBackUserData);

OSErr           doStartDrag             (EventRecord *,RgnHandle,TEHandle);
OSErr           dragTrackingHandler     (DragTrackingMessage,WindowRef,void *,DragRef);
SInt16          doGetOffset             (Point,TEHandle);
SInt16          doIsOffsetAtLineStart   (SInt16,TEHandle);
void            doDrawCaret             (SInt16,TEHandle);
SInt16          doGetLine               (SInt16,TEHandle);

OSErr           dragReceiveHandler      (WindowRef,void *,DragRef);
Boolean         doIsWhiteSpaceAtOffset  (SInt16,TEHandle);
Boolean         doIsWhiteSpace          (char);
char            doGetCharAtOffset       (SInt16,TEHandle);
SInt16          doInsertTextAtOffset    (SInt16,Ptr,SInt32,TEHandle);
Boolean         doSavePreInsertionText  (docStructurePointer);
void            doUndoRedoDrag          (WindowRef);

// ****************************************************************************************
// Drag.c
```

```
// ****************************************************************************************

// ..................................................................... includes

#include "Drag.h"

// ................................................................... global variables

ControlActionUPP       gScrollActionFunctionUPP;
TEClickLoopUPP         gCustomClickLoopUPP;
DragTrackingHandlerUPP gDragTrackingHandlerUPP;
DragReceiveHandlerUPP  gDragReceiveHandlerUPP;
Boolean                gRunningOnX             = false;
SInt16                 gNumberOfWindows        = 0;
SInt16                 gOldControlValue;
Boolean                gEnableDragUndoRedoItem = false;
Boolean                gUndoFlag;

// ********************************************************************************** main

void  main(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  EventTypeSpec applicationEvents[] = { {  kEventClassApplication, kEventAppActivated    } ,
                                        {  kEventClassApplication, kEventAppDeactivated  } ,
                                        {  kEventClassCommand,     kEventProcessCommand  } ,
                                        {  kEventClassMenu,        kEventMenuEnableItems } ,
                                        {  kEventClassMouse,       kEventMouseMoved      }  } ;

// ..................................................................... do preliminaries

doPreliminaries();

// ..................................................... create universal procedure pointers

gScrollActionFunctionUPP = NewControlActionUPP((ControlActionProcPtr) scrollActionFunction);
gCustomClickLoopUPP      = NewTEClickLoopUPP((TEClickLoopProcPtr) customClickLoop);

gDragTrackingHandlerUPP  = NewDragTrackingHandlerUPP((DragTrackingHandlerProcPtr)
                                                      dragTrackingHandler);
gDragReceiveHandlerUPP   = NewDragReceiveHandlerUPP((DragReceiveHandlerProcPtr)
                                                    dragReceiveHandler);
// ..................................................... set up menu bar and menus

menubarHdl = GetNewMBar(rMenubar);
if(menubarHdl == NULL)
```

```
  doErrorAlert(eMenuBar);
SetMenuBar(menubarHdl);
DrawMenuBar();

Gestalt(gestaltMenuMgrAttr,&response);
if(response & gestaltMenuMgrAquaLayoutMask)
{
  menuRef = GetMenuRef(mFile);
  if(menuRef != NULL)
  {
    DeleteMenuItem(menuRef,iQuit);
    DeleteMenuItem(menuRef,iQuit - 1);
  }

  gRunningOnX = true;
}
else
{
  menuRef = GetMenuRef(mFile);
  if(menuRef != NULL)
    SetMenuItemCommandID(menuRef,iQuit,kHICommandQuit);
}

// ..................................................... install application event handler

InstallApplicationEventHandler(NewEventHandlerUPP((EventHandlerProcPtr) appEventHandler),
                               GetEventTypeCount(applicationEvents),applicationEvents,
                               0,NULL);

// ........................................................................ install a timer

InstallEventLoopTimer(GetCurrentEventLoop(),0,TicksToEventTime(GetCaretTime()),
                      NewEventLoopTimerUPP((EventLoopTimerProcPtr) doIdle),NULL,
                      NULL);

// ..................................................................... open window

doNewDocWindow();

// ..................................................... run application event loop

RunApplicationEventLoop();
}

// ********************************************************************** doPreliminaries

void  doPreliminaries(void)
{
```

```
  MoreMasterPointers(192);
  InitCursor();
}
// ************************************************************************** appEventHandler

OSStatus  appEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                          void * userData)
{
  OSStatus      result = eventNotHandledErr;
  UInt32        eventClass;
  UInt32        eventKind;
  HICommand     hiCommand;
  MenuID        menuID;
  MenuItemIndex menuItem;
  WindowClass   windowClass;

  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);

  switch(eventClass)
  {
    case kEventClassApplication:
      if(eventKind == kEventAppActivated)
        SetThemeCursor(kThemeArrowCursor);
      break;

    case kEventClassCommand:
      if(eventKind == kEventProcessCommand)
      {
        GetEventParameter(eventRef,kEventParamDirectObject,typeHICommand,NULL,
                          sizeof(HICommand),NULL,&hiCommand);
        menuID = GetMenuID(hiCommand.menu.menuRef);
        menuItem = hiCommand.menu.menuItemIndex;
        if((hiCommand.commandID != kHICommandQuit) &&
           (menuID >= mAppleApplication && menuID <= mEdit))
        {
          doMenuChoice(menuID,menuItem);
          result = noErr;
        }
      }
      break;

    case kEventClassMenu:
      if(eventKind == kEventMenuEnableItems)
      {
        GetWindowClass(FrontWindow(),&windowClass);
        if(windowClass == kDocumentWindowClass)
          doAdjustMenus();
```

```
        result = noErr;
      }
      break;

    case kEventClassMouse:
      if(eventKind == kEventMouseMoved)
      {
        GetWindowClass(FrontWindow(),&windowClass);
        if(windowClass == kDocumentWindowClass)
          doAdjustCursor(FrontWindow());
        result = noErr;
      }
      break;
  }

  return result;
}

// ********************************************************************** windowEventHandler

OSStatus  windowEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                             void* userData)
{
  OSStatus        result = eventNotHandledErr;
  UInt32          eventClass;
  UInt32          eventKind;
  WindowRef       windowRef;
  UInt32          modifiers;
  Point           mouseLocation;
  Boolean         shiftKeyDown = false;
  EventRecord     eventRecord;
  ControlRef      controlRef;
  ControlPartCode controlPartCode;
  SInt8           charCode;

  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);

  switch(eventClass)
  {
    case kEventClassWindow:                                          // event class window
      GetEventParameter(eventRef,kEventParamDirectObject,typeWindowRef,NULL,sizeof(windowRef),
                        NULL,&windowRef);
      switch(eventKind)
      {
        case kEventWindowDrawContent:
          doDrawContent(windowRef);
          result = noErr;
```

```
      break;

    case kEventWindowActivated:
      doActivateDeactivate(windowRef,true);
      result = noErr;
      break;

    case kEventWindowDeactivated:
      doActivateDeactivate(windowRef,false);
      result = noErr;
      break;

    case kEventWindowClickContentRgn:
      SetPortWindowPort(FrontWindow());
      GetMouse(&mouseLocation);
      GetEventParameter(eventRef,kEventParamKeyModifiers,typeUInt32,NULL,
                        sizeof(modifiers),NULL,&modifiers);
      if(modifiers & shiftKey)
        shiftKeyDown = true;
      ConvertEventRefToEventRecord(eventRef,&eventRecord);
      doInContent(mouseLocation,&eventRecord,shiftKeyDown);
      result = noErr;
      break;

    case kEventWindowClose:
      doCloseWindow(windowRef);
      result = noErr;
      break;
  }
  break;

case kEventClassMouse:                                          // event class mouse
  switch(eventKind)
  {
    case kEventMouseDown:
      GetEventParameter(eventRef,kEventParamMouseLocation,typeQDPoint,NULL,
                        sizeof(mouseLocation),NULL,&mouseLocation);
      SetPortWindowPort(FrontWindow());
      GlobalToLocal(&mouseLocation);
      controlRef = FindControlUnderMouse(mouseLocation,FrontWindow(),&controlPartCode);
      if(controlRef)
      {
        gOldControlValue = GetControlValue(controlRef);
        TrackControl(controlRef,mouseLocation,gScrollActionFunctionUPP);
        result = noErr;
      }
    break;
```

```
      }
      break;

    case kEventClassKeyboard:                                       // event class keyboard
      switch(eventKind)
      {
        case kEventRawKeyDown:
        case kEventRawKeyRepeat:
          GetEventParameter(eventRef,kEventParamKeyMacCharCodes,typeChar,NULL,
                            sizeof(charCode),NULL,&charCode);
          GetEventParameter(eventRef,kEventParamKeyModifiers,typeUInt32,NULL,
                            sizeof(modifiers),NULL,&modifiers);
          if((modifiers & cmdKey) == 0)
            doKeyEvent(charCode);
          result = noErr;
          break;
      }
      break;
  }

  return result;
}
// ********************************************************************************** doIdle

void  doIdle(void)
{
  WindowRef           windowRef;
  docStructurePointer docStrucPtr;

  windowRef = FrontWindow();
  if(GetWindowKind(windowRef) == kApplicationWindowKind)
  {
    docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));
    if(docStrucPtr != NULL)
      TEIdle(docStrucPtr->textEditStrucHdl);
  }
}

// ********************************************************************************** doKeyEvent

void  doKeyEvent(SInt8 charCode)
{
  WindowRef           windowRef;
  docStructurePointer docStrucPtr;
  TEHandle            textEditStrucHdl;
  SInt16              selectionLength;

  if(charCode <= kEscape && charCode != kBackSpace && charCode != kReturn)
```

```
    return;

  windowRef = FrontWindow();
  docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));
  textEditStrucHdl = docStrucPtr->textEditStrucHdl;

  gEnableDragUndoRedoItem = false;

  if(charCode == kTab)
  {
    // Do tab key handling here if required.
  }
  else if(charCode == kForwardDelete)
  {
    selectionLength = doGetSelectLength(textEditStrucHdl);
    if(selectionLength == 0)
      (*textEditStrucHdl)->selEnd += 1;
    TEDelete(textEditStrucHdl);
    doAdjustScrollbar(windowRef);
  }
  else
  {
    selectionLength = doGetSelectLength(textEditStrucHdl);
    if(((*textEditStrucHdl)->teLength - selectionLength + 1) < kMaxTELength)
    {
      TEKey(charCode,textEditStrucHdl);
      doAdjustScrollbar(windowRef);
    }
    else
      doErrorAlert(eExceedChara);
  }
}

// ********************************************************************** scrollActionFunction

void  scrollActionFunction(ControlRef controlRef,SInt16 partCode)
{
  WindowRef           windowRef;
  docStructurePointer docStrucPtr;
  TEHandle            textEditStrucHdl;
  SInt16              linesToScroll;
  SInt16              controlValue, controlMax;

  windowRef = GetControlOwner(controlRef);
  docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));;
  textEditStrucHdl = docStrucPtr->textEditStrucHdl;

  controlValue = GetControlValue(controlRef);
```

```
  controlMax = GetControlMaximum(controlRef);

  if(partCode)
  {
    if(partCode != kControlIndicatorPart)
    {
      switch(partCode)
      {
        case kControlUpButtonPart:
        case kControlDownButtonPart:
          linesToScroll = 1;
          break;

        case kControlPageUpPart:
        case kControlPageDownPart:
          linesToScroll = (((*textEditStrucHdl)->viewRect.bottom -
                           (*textEditStrucHdl)->viewRect.top) /
                           (*textEditStrucHdl)->lineHeight) - 1;
          break;
      }

      if((partCode == kControlDownButtonPart) || (partCode == kControlPageDownPart))

        linesToScroll = -linesToScroll;
      linesToScroll = controlValue - linesToScroll;
      if(linesToScroll < 0)
        linesToScroll = 0;
      else if(linesToScroll > controlMax)
        linesToScroll = controlMax;

      SetControlValue(controlRef,linesToScroll);

      linesToScroll = controlValue - linesToScroll;
    }
    else
    {
      linesToScroll = gOldControlValue - controlValue;
      gOldControlValue = controlValue;
    }

    if(linesToScroll != 0)
      TEScroll(0,linesToScroll * (*textEditStrucHdl)->lineHeight,textEditStrucHdl);
  }
}

// ********************************************************************************* doInContent

void  doInContent(Point mouseLocation,EventRecord * eventStrucPtr,Boolean shiftKeyDown)
```

```
{
  WindowRef           windowRef;
  docStructurePointer docStrucPtr;
  TEHandle            textEditStrucHdl;
  RgnHandle           hiliteRgn;
  OSErr               osError;

  windowRef = FrontWindow();
  docStrucPtr = (docStructurePointer) GetWRefCon(windowRef);
  textEditStrucHdl = docStrucPtr->textEditStrucHdl;

  if(PtInRect(mouseLocation,&(*textEditStrucHdl)->viewRect))
  {
    hiliteRgn = NewRgn();

    TEGetHiliteRgn(hiliteRgn,textEditStrucHdl);

    if(!EmptyRgn(hiliteRgn) && PtInRgn(mouseLocation,hiliteRgn))
    {
      if(WaitMouseMoved(mouseLocation))
      {
        osError = doStartDrag(eventStrucPtr,hiliteRgn,textEditStrucHdl);
        if(osError != noErr)
          doErrorAlert(eDrag);
      }
    }
    else
    {
      TEClick(mouseLocation,shiftKeyDown,textEditStrucHdl);
      gEnableDragUndoRedoItem = false;
    }

    DisposeRgn(hiliteRgn);
  }
}
// ********************************************************************************* doDrawContent

void  doDrawContent(WindowRef windowRef)
{
  docStructurePointer docStrucPtr;
  TEHandle            textEditStrucHdl;
  GrafPtr             oldPort;
  RgnHandle           visibleRegionHdl = NewRgn();
  Rect                portRect;

  docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));
  textEditStrucHdl = docStrucPtr->textEditStrucHdl;
```

```
  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

  GetPortVisibleRegion(GetWindowPort(windowRef),visibleRegionHdl);
  EraseRgn(visibleRegionHdl);

  UpdateControls(windowRef,visibleRegionHdl);

  GetWindowPortBounds(windowRef,&portRect);
  TEUpdate(&(*textEditStrucHdl)->viewRect,textEditStrucHdl);

  DisposeRgn(visibleRegionHdl);
  SetPort(oldPort);
}

// ************************************************************************ doActivateDocWindow

void  doActivateDeactivate(WindowRef windowRef,Boolean becomingActive)
{
  docStructurePointer docStrucPtr;
  TEHandle            textEditStrucHdl;

  docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));
  textEditStrucHdl = docStrucPtr->textEditStrucHdl;

  if(becomingActive)
  {
    SetPortWindowPort(windowRef);

    (*textEditStrucHdl)->viewRect.bottom = ((((*textEditStrucHdl)->viewRect.bottom -
                                          (*textEditStrucHdl)->viewRect.top) /
                                          (*textEditStrucHdl)->lineHeight) *
                                          (*textEditStrucHdl)->lineHeight) +
                                          (*textEditStrucHdl)->viewRect.top;
    (*textEditStrucHdl)->destRect.bottom = (*textEditStrucHdl)->viewRect.bottom;

    TEActivate(textEditStrucHdl);
    ActivateControl(docStrucPtr->vScrollbarRef);
    doAdjustScrollbar(windowRef);
    doAdjustCursor(windowRef);
  }
  else
  {
    TEDeactivate(textEditStrucHdl);
    DeactivateControl(docStrucPtr->vScrollbarRef);
  }
}
```

```
// ************************************************************************** doNewDocWindow

WindowRef  doNewDocWindow(void)
{
  WindowRef           windowRef;
  OSStatus            osError;
  Rect                contentRect = {  100,100,400,595 } ;
  WindowAttributes    attributes  = kWindowStandardHandlerAttribute |
                                    kWindowStandardDocumentAttributes;
  docStructurePointer docStrucPtr;
  Rect                portRect, destAndViewRect;
  EventTypeSpec   windowEvents[] = {  {  kEventClassWindow,   kEventWindowDrawContent     } ,
                                      {  kEventClassWindow,   kEventWindowActivated       } ,
                                      {  kEventClassWindow,   kEventWindowDeactivated     } ,
                                      {  kEventClassWindow,   kEventWindowClickContentRgn } ,
                                      {  kEventClassWindow,   kEventWindowClose           } ,
                                      {  kEventClassMouse,    kEventMouseDown             } ,
                                      {  kEventClassKeyboard, kEventRawKeyDown            } ,
                                      {  kEventClassKeyboard, kEventRawKeyRepeat          }  } ;

  osError = CreateNewWindow(kDocumentWindowClass,attributes,&contentRect,&windowRef);
  if(osError != noErr)
  {
    doErrorAlert(eWindow);
    return NULL;
  }

  ChangeWindowAttributes(windowRef,0,kWindowResizableAttribute);
  RepositionWindow(windowRef,NULL,kWindowCascadeOnMainScreen);
  SetWTitle(windowRef,"\puntitled");
  SetPortWindowPort(windowRef);
  TextSize(10);
  if(!(docStrucPtr = (docStructurePointer) NewPtr(sizeof(docStructure))))
  {
    doErrorAlert(eDocStructure);
    return NULL;
  }

  SetWRefCon(windowRef,(SInt32) docStrucPtr);
  SetWindowProxyCreatorAndType(windowRef,0,'TEXT',kUserDomain);

  InstallWindowEventHandler(windowRef,doGetHandlerUPP(),GetEventTypeCount(windowEvents),
                            windowEvents,0,NULL);

  gNumberOfWindows ++;
  docStrucPtr->windowRef     = windowRef;
  docStrucPtr->windowTouched = false;
  docStrucPtr->preDragText   = NULL;
```

```
  docStrucPtr->vScrollbarRef = GetNewControl(rVScrollbar,windowRef);

  GetWindowPortBounds(windowRef,&portRect);
  destAndViewRect = portRect;
  destAndViewRect.right -= 15;
  InsetRect(&destAndViewRect,2,2);

  if(!(docStrucPtr->textEditStrucHdl = TENew(&destAndViewRect,&destAndViewRect)))
  {
    DisposeWindow(windowRef);
    gNumberOfWindows --;
    DisposePtr((Ptr) docStrucPtr);
    doErrorAlert(eTextEdit);
    return NULL;
  }

  TESetClickLoop(gCustomClickLoopUPP,docStrucPtr->textEditStrucHdl);
  TEAutoView(true,docStrucPtr->textEditStrucHdl);
  TEFeatureFlag(teFOutlineHilite,teBitSet,docStrucPtr->textEditStrucHdl);

  if(osError = InstallTrackingHandler(gDragTrackingHandlerUPP,windowRef,docStrucPtr))
  {
    DisposeWindow(windowRef);
    gNumberOfWindows --;
    DisposePtr((Ptr) docStrucPtr);
    doErrorAlert(eDragHandler);
    return NULL;
  }

  if(osError = InstallReceiveHandler(gDragReceiveHandlerUPP,windowRef,docStrucPtr))
  {
    RemoveTrackingHandler(gDragTrackingHandlerUPP,windowRef);
    DisposeWindow(windowRef);
    gNumberOfWindows --;
    DisposePtr((Ptr) docStrucPtr);
    doErrorAlert(eDragHandler);
    return NULL;
  }


  ShowWindow(windowRef);

  return windowRef;
}

// ********************************************************************** doGetHandlerUPP
```

```
EventHandlerUPP  doGetHandlerUPP(void)
{
  static EventHandlerUPP windowEventHandlerUPP;
  if(windowEventHandlerUPP == NULL)
    windowEventHandlerUPP = NewEventHandlerUPP((EventHandlerProcPtr) windowEventHandler);

  return windowEventHandlerUPP;
}

// ********************************************************************** customClickLoop

Boolean  customClickLoop(void)
{
  WindowRef           windowRef;
  docStructurePointer docStrucPtr;
  TEHandle            textEditStrucHdl;
  GrafPtr             oldPort;
  RgnHandle           oldClip;
  Rect                tempRect, portRect;
  Point               mouseXY;
  SInt16              linesToScroll = 0;

  windowRef = FrontWindow();
  docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));
  textEditStrucHdl = docStrucPtr->textEditStrucHdl;

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);
  oldClip = NewRgn();
  GetClip(oldClip);
  SetRect(&tempRect,-32767,-32767,32767,32767);
  ClipRect(&tempRect);

  GetMouse(&mouseXY);
  GetWindowPortBounds(windowRef,&portRect);

  if(mouseXY.v < portRect.top)
  {
    linesToScroll = 1;
    doSetScrollBarValue(docStrucPtr->vScrollbarRef,&linesToScroll);
    if(linesToScroll != 0)
      TEScroll(0,linesToScroll * ((*textEditStrucHdl)->lineHeight),textEditStrucHdl);
  }
  else if(mouseXY.v > portRect.bottom)
  {
    linesToScroll = -1;
    doSetScrollBarValue(docStrucPtr->vScrollbarRef,&linesToScroll);
    if(linesToScroll != 0)
```

```
      TEScroll(0,linesToScroll * ((*textEditStrucHdl)->lineHeight),textEditStrucHdl);
  }

  SetClip(oldClip);
  DisposeRgn(oldClip);
  SetPort(oldPort);

  return true;
}
// ********************************************************************** doSetScrollBarValue

void  doSetScrollBarValue(ControlRef controlRef,SInt16 *linesToScroll)
{
  SInt16 controlValue, controlMax;

  controlValue = GetControlValue(controlRef);
  controlMax = GetControlMaximum(controlRef);

  *linesToScroll = controlValue - *linesToScroll;
  if(*linesToScroll < 0)
    *linesToScroll = 0;
  else if(*linesToScroll > controlMax)
    *linesToScroll = controlMax;

  SetControlValue(controlRef,*linesToScroll);
  *linesToScroll = controlValue - *linesToScroll;
}

// ************************************************************************** doAdjustMenus

void  doAdjustMenus(void)
{
  MenuRef             fileMenuRef, editMenuRef;
  WindowRef           windowRef;
  docStructurePointer docStrucPtr;
  TEHandle            textEditStrucHdl;
  ScrapRef            scrapRef;
  OSStatus            osError;
  ScrapFlavorFlags    scrapFlavorFlags;

  fileMenuRef = GetMenuRef(mFile);
  editMenuRef = GetMenuRef(mEdit);

  if(gNumberOfWindows > 0)
  {
    windowRef = FrontWindow();
    docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));;
    textEditStrucHdl = docStrucPtr->textEditStrucHdl;
```

```
  EnableMenuItem(fileMenuRef,iClose);

  if(gEnableDragUndoRedoItem)
  {
    EnableMenuItem(editMenuRef,iUndo);
    if(gUndoFlag)
      SetMenuItemText(editMenuRef,iUndo,"\pUndo Drag & Drop");
    else
      SetMenuItemText(editMenuRef,iUndo,"\pRedo Drag & Drop");
  }
  else
  {
    DisableMenuItem(editMenuRef,iUndo);
    SetMenuItemText(editMenuRef,iUndo,"\pRedo Drag & Drop");
  }
  if((*textEditStrucHdl)->selStart < (*textEditStrucHdl)->selEnd)
  {
    EnableMenuItem(editMenuRef,iCut);
    EnableMenuItem(editMenuRef,iCopy);
    EnableMenuItem(editMenuRef,iClear);
  }
  else
  {
    DisableMenuItem(editMenuRef,iCut);
    DisableMenuItem(editMenuRef,iCopy);
    DisableMenuItem(editMenuRef,iClear);
  }

  GetCurrentScrap(&scrapRef);

  osError = GetScrapFlavorFlags(scrapRef,kScrapFlavorTypeText,&scrapFlavorFlags);
  if(osError == noErr)
    EnableMenuItem(editMenuRef,iPaste);
  else
    DisableMenuItem(editMenuRef,iPaste);

  if((*textEditStrucHdl)->teLength > 0)
  {
    EnableMenuItem(fileMenuRef,iSaveAs);
    EnableMenuItem(editMenuRef,iSelectAll);
  }
  else
  {
    DisableMenuItem(fileMenuRef,iSaveAs);
    DisableMenuItem(editMenuRef,iSelectAll);
  }
}
```

```
  else
  {
    DisableMenuItem(fileMenuRef,iClose);
    DisableMenuItem(fileMenuRef,iSaveAs);
    DisableMenuItem(editMenuRef,iClear);
    DisableMenuItem(editMenuRef,iSelectAll);
  }

  DrawMenuBar();
}

// ********************************************************************************* doMenuChoice

void  doMenuChoice(MenuID menuID,MenuItemIndex menuItem)
{
  if(menuID == 0)
    return;

  switch(menuID)
  {
    case mAppleApplication:
      if(menuItem == iAbout)
        SysBeep(10);
      break;
    case mFile:
      doFileMenu(menuItem);
      break;

    case mEdit:
      doEditMenu(menuItem);
      break;
  }
}

// ********************************************************************************** doFileMenu

void  doFileMenu(MenuItemIndex menuItem)
{
  docStructurePointer docStrucPtr;
  TEHandle            textEditStrucHdl;

  switch(menuItem)
  {
    case iNew:
      doNewDocWindow();
      break;

    case iOpen:
```

```
      doOpenCommand();
      doAdjustScrollbar(FrontWindow());
      break;

    case iClose:
      doCloseWindow(FrontWindow());
      break;

    case iSaveAs:
      docStrucPtr = (docStructurePointer) (GetWRefCon(FrontWindow()));
      textEditStrucHdl = docStrucPtr->textEditStrucHdl;
      doSaveAsFile(textEditStrucHdl);
      break;
  }
}

// ********************************************************************************** doEditMenu

void  doEditMenu(MenuItemIndex menuItem)
{
  WindowRef           windowRef;
  docStructurePointer docStrucPtr;
  TEHandle            textEditStrucHdl;
  SInt32              totalSize, contigSize, newSize;
  SInt16              selectionLength;
  ScrapRef            scrapRef;
  Size                sizeOfTextData;

  windowRef = FrontWindow();
  docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));
  textEditStrucHdl = docStrucPtr->textEditStrucHdl;
  switch(menuItem)
  {
    case iUndo:
      doUndoRedoDrag(windowRef);
      break;

    case iCut:
      if(ClearCurrentScrap() == noErr)
      {
        PurgeSpace(&totalSize,&contigSize);
        selectionLength = doGetSelectLength(textEditStrucHdl);
        if(selectionLength > contigSize)
          doErrorAlert(eNoSpaceCut);
        else
        {
          TECut(textEditStrucHdl);
          doAdjustScrollbar(windowRef);
```

```
          if(TEToScrap() != noErr)
            ClearCurrentScrap();
        }
      }
      break;

    case iCopy:
      if(ClearCurrentScrap() == noErr)
      {
        TECopy(textEditStrucHdl);
        if(TEToScrap() != noErr)
          ClearCurrentScrap();
      }
      break;

    case iPaste:
      GetCurrentScrap(&scrapRef);;
      GetScrapFlavorSize(scrapRef,kScrapFlavorTypeText,&sizeOfTextData);
      newSize = (*textEditStrucHdl)->teLength + sizeOfTextData;
      if(newSize > kMaxTELength)
        doErrorAlert(eNoSpacePaste);
      else
      {
        if(TEFromScrap() == noErr)
        {
          TEPaste(textEditStrucHdl);
          doAdjustScrollbar(windowRef);
        }
      }
      break;

    case iClear:
      TEDelete(textEditStrucHdl);
      doAdjustScrollbar(windowRef);
      break;
    case iSelectAll:
      TESetSelect(0,(*textEditStrucHdl)->teLength,textEditStrucHdl);
      break;
  }
}

// ********************************************************************** doGetSelectLength

SInt16  doGetSelectLength(TEHandle textEditStrucHdl)
{
  SInt16 selectionLength;
  selectionLength = (*textEditStrucHdl)->selEnd - (*textEditStrucHdl)->selStart;
  return selectionLength;
```

```
}

// ************************************************************************** doAdjustScrollbar

void  doAdjustScrollbar(WindowRef windowRef)
{
  docStructurePointer docStrucPtr;
  TEHandle            textEditStrucHdl;
  SInt16              numberOfLines, controlMax, controlValue;

  docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));;
  textEditStrucHdl = docStrucPtr->textEditStrucHdl;

  numberOfLines = (*textEditStrucHdl)->nLines;
  if(*(*(*textEditStrucHdl)->hText + (*textEditStrucHdl)->teLength - 1) == kReturn)
    numberOfLines += 1;

  controlMax = numberOfLines - (((*textEditStrucHdl)->viewRect.bottom -
               (*textEditStrucHdl)->viewRect.top) /
               (*textEditStrucHdl)->lineHeight);
  if(controlMax < 0)
    controlMax = 0;
  SetControlMaximum(docStrucPtr->vScrollbarRef,controlMax);

  controlValue = ((*textEditStrucHdl)->viewRect.top - (*textEditStrucHdl)->destRect.top) /
                  (*textEditStrucHdl)->lineHeight;
  if(controlValue < 0)
    controlValue = 0;
  else if(controlValue > controlMax)
    controlValue = controlMax;

  SetControlValue(docStrucPtr->vScrollbarRef,controlValue);

  SetControlViewSize(docStrucPtr->vScrollbarRef,(*textEditStrucHdl)->viewRect.bottom -
                    (*textEditStrucHdl)->viewRect.top);

  TEScroll(0,((*textEditStrucHdl)->viewRect.top - (*textEditStrucHdl)->destRect.top) -
              (GetControlValue(docStrucPtr->vScrollbarRef) *
             (*textEditStrucHdl)->lineHeight),textEditStrucHdl);
}
// ************************************************************************** doAdjustCursor

void  doAdjustCursor(WindowRef windowRef)
{
  GrafPtr             oldPort;
  RgnHandle           arrowRegion, iBeamRegion, hiliteRgn;
  Rect                portRect, cursorRect;
```

```
  docStructurePointer docStrucPtr;
  Point               offset, mouseXY;

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

  arrowRegion = NewRgn();
  iBeamRegion = NewRgn();
  hiliteRgn    = NewRgn();
  SetRectRgn(arrowRegion,-32768,-32768,32766,32766);

  GetWindowPortBounds(windowRef,&portRect);
  cursorRect = portRect;
  cursorRect.right  -= 15;
  LocalToGlobal(&topLeft(cursorRect));
  LocalToGlobal(&botRight(cursorRect));

  RectRgn(iBeamRegion,&cursorRect);
  DiffRgn(arrowRegion,iBeamRegion,arrowRegion);

  docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));
  TEGetHiliteRgn(hiliteRgn,docStrucPtr->textEditStrucHdl);
  LocalToGlobal(&topLeft(portRect));
  offset = topLeft(portRect);
  OffsetRgn(hiliteRgn,offset.h,offset.v);
  DiffRgn(iBeamRegion,hiliteRgn,iBeamRegion);

  GetGlobalMouse(&mouseXY);

  if(PtInRgn(mouseXY,iBeamRegion))
    SetThemeCursor(kThemeIBeamCursor);
  else if(PtInRgn(mouseXY,hiliteRgn))
    SetThemeCursor(kThemeArrowCursor);
  else
    SetThemeCursor(kThemeArrowCursor);

  DisposeRgn(arrowRegion);
  DisposeRgn(iBeamRegion);
  DisposeRgn(hiliteRgn);

  SetPort(oldPort);
}
// ********************************************************************** doCloseWindow

void  doCloseWindow(WindowRef windowRef)
{
  docStructurePointer docStrucPtr;
```

```
  docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));;

  DisposeControl(docStrucPtr->vScrollbarRef);
  TEDispose(docStrucPtr->textEditStrucHdl);
  DisposePtr((Ptr) docStrucPtr);

  if(docStrucPtr->preDragText == NULL)
    DisposeHandle(docStrucPtr->preDragText);

  RemoveTrackingHandler(gDragTrackingHandlerUPP,windowRef);
  RemoveReceiveHandler(gDragReceiveHandlerUPP,windowRef);

  DisposeWindow(windowRef);

  gNumberOfWindows --;
}

// ********************************************************************************** doSaveAsFile

void  doSaveAsFile(TEHandle textEditStrucHdl)
{
  OSErr            osError = noErr;
  NavDialogOptions dialogOptions;
  NavEventUPP      navEventFunctionUPP;
  WindowRef        windowRef;
  OSType           fileType;
  NavReplyRecord   navReplyStruc;
  AEKeyword        theKeyword;
  DescType         actualType;
  FSSpec           fileSpec;
  SInt16           fileRefNum;
  Size             actualSize;
  SInt32           dataLength;
  Handle           editTextHdl;

  osError = NavGetDefaultDialogOptions(&dialogOptions);

  if(osError == noErr)
  {
    windowRef = FrontWindow();

    fileType = 'TEXT';

    navEventFunctionUPP = NewNavEventUPP((NavEventProcPtr) navEventFunction);
    osError = NavPutFile(NULL,&navReplyStruc,&dialogOptions,navEventFunctionUPP,fileType,
                         kFileCreator,NULL);
    DisposeNavEventUPP(navEventFunctionUPP);
    if(navReplyStruc.validRecord && osError == noErr)
```

```
    {
      if((osError = AEGetNthPtr(&(navReplyStruc.selection),1,typeFSS,&theKeyword,
                                &actualType,&fileSpec,sizeof(fileSpec),&actualSize)) == noErr)

      {
        if(!navReplyStruc.replacing)
        {
          osError = FSpCreate(&fileSpec,kFileCreator,fileType,navReplyStruc.keyScript);
          if(osError != noErr)
          {
            NavDisposeReply(&navReplyStruc);
          }
        }

        if(osError == noErr)
          osError = FSpOpenDF(&fileSpec,fsRdWrPerm,&fileRefNum);

        if(osError == noErr)
        {
          SetWTitle(windowRef,fileSpec.name);
          dataLength = (*textEditStrucHdl)->teLength;
          editTextHdl = (*textEditStrucHdl)->hText;
          FSWrite(fileRefNum,&dataLength,*editTextHdl);
        }

        NavCompleteSave(&navReplyStruc,kNavTranslateInPlace);
      }

      NavDisposeReply(&navReplyStruc);
    }
  }
}

// ********************************************************************************* doOpenCommand

void  doOpenCommand(void)
{
  OSErr            osError  = noErr;
  NavDialogOptions dialogOptions;
  NavEventUPP      navEventFunctionUPP;
  NavReplyRecord   navReplyStruc;
  SInt32           index, count;
  AEKeyword        theKeyword;
  DescType         actualType;
  FSSpec           fileSpec;
  Size             actualSize;
  FInfo            fileInfo;
```

```
  osError = NavGetDefaultDialogOptions(&dialogOptions);
  if(osError == noErr)
  {
    navEventFunctionUPP = NewNavEventUPP((NavEventProcPtr) navEventFunction);
    osError = NavGetFile(NULL,&navReplyStruc,&dialogOptions,navEventFunctionUPP,NULL,NULL,
                         NULL,NULL);
    DisposeNavEventUPP(navEventFunctionUPP);

    if(osError == noErr && navReplyStruc.validRecord)
    {
      if(osError == noErr)
      {
        if(osError == noErr)
        {
          osError = AECountItems(&(navReplyStruc.selection),&count);

          for(index=1;index<=count;index++)
          {
            osError = AEGetNthPtr(&(navReplyStruc.selection),index,typeFSS,&theKeyword,
                                  &actualType,&fileSpec,sizeof(fileSpec),&actualSize);
            {
              if((osError = FSpGetFInfo(&fileSpec,&fileInfo)) == noErr)
                doOpenFile(fileSpec);
            }
          }
        }
      }

      NavDisposeReply(&navReplyStruc);
    }
  }
}

// ************************************************************************** doOpenFile

void  doOpenFile(FSSpec fileSpec)
{
  WindowRef           windowRef;
  docStructurePointer docStrucPtr;
  TEHandle            textEditStrucHdl;
  SInt16              fileRefNum;
  SInt32              textLength;
  Handle              textBuffer;

  if((windowRef = doNewDocWindow()) == NULL)
    return;

  docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));
```

```
  textEditStrucHdl = docStrucPtr->textEditStrucHdl;

  SetWTitle(windowRef,fileSpec.name);

  FSpOpenDF(&fileSpec,fsCurPerm,&fileRefNum);
  SetFPos(fileRefNum,fsFromStart,0);
  GetEOF(fileRefNum,&textLength);

  if(textLength > 32767)
    textLength = 32767;

  textBuffer = NewHandle((Size) textLength);

  FSRead(fileRefNum,&textLength,*textBuffer);

  MoveHHi(textBuffer);
  HLock(textBuffer);

  TESetText(*textBuffer,textLength,textEditStrucHdl);

  HUnlock(textBuffer);
  DisposeHandle(textBuffer);

  FSClose(fileRefNum);

  (*textEditStrucHdl)->selStart = 0;
  (*textEditStrucHdl)->selEnd = 0;

  doDrawContent(windowRef);
}

// ************************************************************************** doErrorAlert

void  doErrorAlert(SInt16 errorCode)
{
  Str255 errorString;
  SInt16 itemHit;

  GetIndString(errorString,rErrorStrings,errorCode);

  if(errorCode < eWindow)
  {
    StandardAlert(kAlertStopAlert,errorString,NULL,NULL,&itemHit);
    ExitToShell();
  }
  else
  {
    StandardAlert(kAlertCautionAlert,errorString,NULL,NULL,&itemHit);
```

```
  }
}

// ********************************************************************** navEventFunction

void  navEventFunction(NavEventCallbackMessage callBackSelector,NavCBRecPtr callBackParms,
                       NavCallBackUserData callBackUD)
{
}
// ****************************************************************************************
// StartAndTrackDrag.c
// ****************************************************************************************

// ..................................................................................... includes

#include "Drag.h"

// ..................................................................... global variables

Boolean gCursorInContent, gCanAcceptItems, gCaretShowFlag;
SInt16  gInsertPosition, gLastOffset, gCaretOffset;
UInt32  gSystemCaretTime, gCaretStartTime;

extern Boolean gRunningOnX;

// ************************************************************************** doStartDrag

OSErr  doStartDrag(EventRecord *eventStrucPtr,RgnHandle hiliteRgnHdl,TEHandle textEditStrucHdl)
{
  OSErr         osError;
  DragReference dragRef;
  Rect          originalHiliteRect, zeroedHiliteRect;
  RgnHandle     maskRgnHdl;
  Point         offsetPoint;
  QDErr         qdError;
  CGrafPtr      savedPortPtr;
  GDHandle      saveDeviceHdl;
  GWorldPtr     dragGWorldPtr = NULL;
  PixMapHandle  dragPixMapHdl, windPixMapHdl;
  RgnHandle     dragRgnHdl, tempRgnHdl;

  // ..................................................................... create new drag

  if(osError = NewDrag(&dragRef))
    return osError;

  // ..................................................................... add 'TEXT' flavor
```

```
osError = AddDragItemFlavor(dragRef,1,'TEXT',
                            (*(*textEditStrucHdl)->hText) + (*textEditStrucHdl)->selStart,
                            (*textEditStrucHdl)->selEnd - (*textEditStrucHdl)->selStart,0);

// ................................... get and set drag image for translucent drag and drop

if(!gRunningOnX)
{
  maskRgnHdl =  dragRgnHdl = tempRgnHdl = NULL;

  GetRegionBounds(hiliteRgnHdl,&originalHiliteRect);
  zeroedHiliteRect = originalHiliteRect;
  OffsetRect(&zeroedHiliteRect,-originalHiliteRect.left,-originalHiliteRect.top);

  GetGWorld(&savedPortPtr,&saveDeviceHdl);
  qdError = NewGWorld(&dragGWorldPtr,8,&zeroedHiliteRect,NULL,NULL,0);
  if(dragGWorldPtr != NULL && qdError == noErr)
  {
    SetGWorld(dragGWorldPtr,NULL);
    EraseRect(&zeroedHiliteRect);

    dragPixMapHdl = GetGWorldPixMap(dragGWorldPtr);
    LockPixels(dragPixMapHdl);
    windPixMapHdl = GetGWorldPixMap(savedPortPtr);

    CopyBits((BitMap *) *windPixMapHdl,(BitMap *) *dragPixMapHdl,
             &originalHiliteRect,&zeroedHiliteRect,srcCopy,NULL);

    UnlockPixels(dragPixMapHdl);
    SetGWorld(savedPortPtr,saveDeviceHdl);

    maskRgnHdl = NewRgn();
    if(maskRgnHdl != NULL)
    {
      CopyRgn(hiliteRgnHdl,maskRgnHdl);
      OffsetRgn(maskRgnHdl,-originalHiliteRect.left,-originalHiliteRect.top);

      SetPt(&offsetPoint,originalHiliteRect.left,originalHiliteRect.top);
      LocalToGlobal(&offsetPoint);

      SetDragImage(dragRef,dragPixMapHdl,maskRgnHdl,offsetPoint,kDragStandardTranslucency);
    }
  }
}

// ..................................................................... get drag region
```

```
  dragRgnHdl = NewRgn();
  if(dragRgnHdl == NULL)
    return MemError();

  CopyRgn(hiliteRgnHdl,dragRgnHdl);
  SetPt(&offsetPoint,0,0);
  LocalToGlobal(&offsetPoint);
  OffsetRgn(dragRgnHdl,offsetPoint.h,offsetPoint.v);

  tempRgnHdl = NewRgn();
  if(tempRgnHdl == NULL)
    return MemError();

  CopyRgn(dragRgnHdl,tempRgnHdl);
  InsetRgn(tempRgnHdl,1,1);
  DiffRgn(dragRgnHdl,tempRgnHdl,dragRgnHdl);
  DisposeRgn(tempRgnHdl);

  // ..................................................................... perform the drag

  osError = TrackDrag(dragRef,eventStrucPtr,dragRgnHdl);
  if(osError != noErr && osError != userCanceledErr)
    return osError;
  if(dragRef)       DisposeDrag(dragRef);
  if(maskRgnHdl)    DisposeRgn(maskRgnHdl);
  if(dragGWorldPtr) DisposeGWorld(dragGWorldPtr);
  if(dragRgnHdl)    DisposeRgn(dragRgnHdl);
  if(tempRgnHdl)    DisposeRgn(tempRgnHdl);

  return noErr;
}

// ********************************************************************** dragTrackingHandler

OSErr dragTrackingHandler(DragTrackingMessage trackingMessage,WindowRef windowRef,
                          void *handlerRefCon,DragRef dragRef)
{
  docStructurePointer docStrucPtr;
  DragAttributes      dragAttributes;
  UInt32              theTime;
  UInt16              numberOfDragItems, index;
  ItemReference       itemRef;
  OSErr               result;
  FlavorFlags         flavorFlags;
  Point               mousePt, localMousePt;
  RgnHandle           windowHiliteRgn;
  Rect                correctedViewRect;
  SInt16              theOffset;
```

```
if((trackingMessage != kDragTrackingEnterHandler) && !gCanAcceptItems)
  return noErr;

docStrucPtr = (docStructurePointer) handlerRefCon;

GetDragAttributes(dragRef,&dragAttributes);
gSystemCaretTime = GetCaretTime();
theTime = TickCount();

switch(trackingMessage)
{
  // ..................................................................... enter handler

  case kDragTrackingEnterHandler:
    gCanAcceptItems = true;

    CountDragItems(dragRef,&numberOfDragItems);

    for(index=1;index <= numberOfDragItems;index++)
    {
      GetDragItemReferenceNumber(dragRef,index,&itemRef);
      result = GetFlavorFlags(dragRef,itemRef,'TEXT',&flavorFlags);
      if(result != noErr)
      {
        gCanAcceptItems = false;
        break;
      }
    }
    break;
    // ..................................................................... enter window

  case kDragTrackingEnterWindow:
    gCaretStartTime  = theTime;
    gCaretOffset     = gLastOffset = -1;
    gCaretShowFlag   = true;
    gCursorInContent = false;
    break;

  // ..................................................................... in window

  case kDragTrackingInWindow:

    GetDragMouse(dragRef,&mousePt,NULL);
    localMousePt = mousePt;
    GlobalToLocal(&localMousePt);

    if(dragAttributes & kDragHasLeftSenderWindow)
```

```
{
  if(PtInRect(localMousePt,&(**(docStrucPtr->textEditStrucHdl)).viewRect))
  {
    if(!gCursorInContent)
    {
      windowHiliteRgn = NewRgn();
      correctedViewRect = (**(docStrucPtr->textEditStrucHdl)).viewRect;
      InsetRect(&correctedViewRect,-2,-2);
      RectRgn(windowHiliteRgn,&correctedViewRect);
      ShowDragHilite(dragRef,windowHiliteRgn,true);
      DisposeRgn(windowHiliteRgn);
    }
    gCursorInContent = true;
  }
  else
  {
    if(gCursorInContent)
      HideDragHilite(dragRef);
    gCursorInContent = false;
  }
}

// ... ... ... ... ... ... ... ... ... ... ... ... start caret drawing stuff, first get the offset into the text

theOffset = doGetOffset(mousePt,docStrucPtr->textEditStrucHdl);

// ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...  if in sender window, defeat caret drawing in selection

if(dragAttributes & kDragInsideSenderWindow)
{
  if((theOffset >= (*(docStrucPtr->textEditStrucHdl))->selStart) &&
     (theOffset <= (*(docStrucPtr->textEditStrucHdl))->selEnd))
  {
    theOffset = -1;
  }
}
// ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... save the offset to a global for use by dragReceiveHandler

gInsertPosition = theOffset;

// ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... if offset has changed, reset the caret flashing timer

if(theOffset != gLastOffset)
{
  gCaretStartTime = theTime;
  gCaretShowFlag = true;
}
```

```
  gLastOffset = theOffset;

  // ... ... ... ... if caret-flashing interval has elapsed, toggle caret "show" flag, reset timer

  if(theTime - gCaretStartTime > gSystemCaretTime)
  {
    gCaretShowFlag = !gCaretShowFlag;
    gCaretStartTime = theTime;
  }

  // ... ... ... ... ... ... ... ... ... ... if caret "show" flag is off, set variable to defeat caret drawing

  if(!gCaretShowFlag)
    theOffset = -1;

  // ... ... ... ... if offset has changed, erase previous caret, draw new caret at current offset
  if(theOffset != gCaretOffset)
  {

    // ... ... ... ... ... ... ... if first pass this window, don't erase, otherwise erase at old offset

    if(gCaretOffset != -1)
      doDrawCaret(gCaretOffset,docStrucPtr->textEditStrucHdl);
    // ... ... ... ... ... ... ...  ... ... ...  ... ... ...  if "show" flag says show, draw caret at current offset

    if(theOffset != -1)
      doDrawCaret(theOffset,docStrucPtr->textEditStrucHdl);
  }

  gCaretOffset = theOffset;

  break;

// ..................................................................... leave window

case kDragTrackingLeaveWindow:

  if(gCaretOffset != -1)
  {
    doDrawCaret(gCaretOffset,docStrucPtr->textEditStrucHdl);
    gCaretOffset = -1;
  }
  if(gCursorInContent && dragAttributes & kDragHasLeftSenderWindow)
    HideDragHilite(dragRef);

  break;

// ..................................................................... leave handler
```

```
    case kDragTrackingLeaveHandler:
      break;
  }

  return noErr;
}

// ********************************************************************************** doGetOffset

SInt16  doGetOffset(Point mousePt,TEHandle textEditStrucHdl)
{
  WindowRef windowRef;
  SInt16    theOffset;
  Point     thePoint;

  theOffset = -1;

  if(FindWindow(mousePt,&windowRef) == inContent)
  {
    SetPortWindowPort(windowRef);
    GlobalToLocal(&mousePt);

    if(PtInRect(mousePt,&((*textEditStrucHdl)->viewRect)))
    {
      theOffset = TEGetOffset(mousePt,textEditStrucHdl);
      thePoint  = TEGetPoint(theOffset - 1,textEditStrucHdl);

      if((theOffset) &&
         (doIsOffsetAtLineStart(theOffset,textEditStrucHdl)) &&
         ((*(*textEditStrucHdl)->hText)[theOffset - 1] != 0x0D) &&
         (thePoint.h < mousePt.h))
      {
        theOffset--;
      }
    }
  }

  return theOffset;
}

// ********************************************************************** doIsOffsetAtLineStart

SInt16  doIsOffsetAtLineStart(SInt16 offset,TEHandle textEditStrucHdl)
{
  SInt16 line = 0;
  if((*textEditStrucHdl)->teLength == 0)
    return(true);
```

```
  if(offset >= (*textEditStrucHdl)->teLength)
    return((*((*textEditStrucHdl)->hText))[(*textEditStrucHdl)->teLength - 1] == 0x0D);

  while((*textEditStrucHdl)->lineStarts[line] < offset)
    line++;

  return ((*textEditStrucHdl)->lineStarts[line] == offset);
}

// ************************************************************************** doDrawCaret

void  doDrawCaret(SInt16 theOffset,TEHandle textEditStrucHdl)
{
  Point  thePoint;
  SInt16 theLine, lineHeight;

  thePoint = TEGetPoint(theOffset,textEditStrucHdl);
  theLine  = doGetLine(theOffset,textEditStrucHdl);

  if((theOffset == (*textEditStrucHdl)->teLength) &&
      (*((*textEditStrucHdl)->hText))[(*textEditStrucHdl)->teLength - 1] == 0x0D)
  {

    thePoint.v += TEGetHeight(theLine,theLine,textEditStrucHdl);
  }

  PenMode(patXor);
  lineHeight = TEGetHeight(theLine,theLine,textEditStrucHdl);
  MoveTo(thePoint.h - 1,thePoint.v - 1);
  Line(0,1 - lineHeight);

  PenNormal();
}

// ************************************************************************** doGetLine

SInt16  doGetLine(SInt16 theOffset,TEHandle textEditStrucHdl)
{
  SInt16 theLine = 0;

  if(theOffset > (*textEditStrucHdl)->teLength)
    return ((*textEditStrucHdl)->nLines);

  while((*textEditStrucHdl)->lineStarts[theLine] < theOffset)
    theLine++;

  return theLine;
```

```
}
// ************************************************************************************************
// ReceiveAndUndoDrag.c
// ************************************************************************************************

// ..................................................................................... includes

#include "Drag.h"

// ................................................................................ global variables

extern Boolean gEnableDragUndoRedoItem;
extern Boolean gUndoFlag;
extern Boolean gCanAcceptItems;
extern SInt16  gInsertPosition, gCaretOffset;

// ********************************************************************** dragReceiveHandler

OSErr  dragReceiveHandler(WindowRef windowRef,void *handlerRefCon,DragRef dragRef)
{
  docStructurePointer docStrucPtr;
  TEHandle            textEditStrucHdl;
  SInt32              totalTextStart;
  Size                totalTextSize;
  Boolean             wasActive, moveText, gotUndoMemory = false;
  DragAttributes      dragAttributes;
  SInt16              mouseDownModifiers, mouseUpModifiers, selStart, selEnd;
  UInt16              numberOfDragItems, index;
  ItemReference       itemReference;
  OSErr               osError;
  Size                textSize;
  Ptr                 textDataPtr;
  SInt32              additionalChars;

  if((!gCanAcceptItems) || (gInsertPosition == -1))
    return dragNotAcceptedErr;

  docStrucPtr = (docStructurePointer) handlerRefCon;
  textEditStrucHdl = docStrucPtr->textEditStrucHdl;

  // ... set graphics port to this window's port and, if necessary, activate text edit structure

  SetPortWindowPort(windowRef);

  wasActive = (*textEditStrucHdl)->active != 0;
  if(!wasActive)
    TEActivate(textEditStrucHdl);
```

```
// ........................................... get drag attributes and keyboard modifiers

GetDragAttributes(dragRef,&dragAttributes);
GetDragModifiers(dragRef,0L,&mouseDownModifiers,&mouseUpModifiers);
// ... ... ...  in case their are multiple items, save first insertion point for later TESetSelect

totalTextStart = gInsertPosition;
totalTextSize = 0;

// ... ... for all items in drag, get 'TEXT' data, insert into this window's text edit structure

CountDragItems(dragRef,&numberOfDragItems);

for(index=1;index <= numberOfDragItems;index++)
{
  GetDragItemReferenceNumber(dragRef,index,&itemReference);

  osError = GetFlavorDataSize(dragRef,itemReference,'TEXT',&textSize);
  if(osError == noErr)
  {
    // ...  if addition of drag to the text edit structure would exceed TextEdit limit, return

    if(((*textEditStrucHdl)->teLength + textSize) > kMaxTELength)
      return dragNotAcceptedErr;

    // ... ... ... ... ... ... ... ... ... ... ... ... ... create nonrelocatable block and get the 'TEXT' data into it

    textDataPtr = NewPtr(textSize);
    if(textDataPtr == NULL)
      return dragNotAcceptedErr;

    GetFlavorData(dragRef,itemReference,'TEXT',textDataPtr,&textSize,0);

    // ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...  if caret or highlighting is on screen, remove it

    if(gCaretOffset != -1)
    {
      doDrawCaret(gCaretOffset,textEditStrucHdl);
      gCaretOffset = -1;
    }

    if(dragAttributes & kDragHasLeftSenderWindow)
      HideDragHilite(dragRef);

    // save current text and selection start/end for Undo, and set Redo/Undo menu item flags

    if(dragAttributes & kDragInsideSenderWindow)
```

```
      {
        gotUndoMemory = doSavePreInsertionText(docStrucPtr);
        if(gotUndoMemory)
        {
          gEnableDragUndoRedoItem = true;
          gUndoFlag = true;
        }
      }
      else
        gEnableDragUndoRedoItem = false;
      // ... ... ... ... ... if in sender window, ensure selected text is deleted if option key not down

      moveText = (dragAttributes & kDragInsideSenderWindow) &&
                 (!((mouseDownModifiers & optionKey) | (mouseUpModifiers & optionKey)));

      if(moveText)
      {
        selStart = (*textEditStrucHdl)->selStart;
        selEnd   = (*textEditStrucHdl)->selEnd;

        // ... ... ... ... ...  extend selection by one chara if space charas just before and just after

        if(doIsWhiteSpaceAtOffset(selStart - 1,textEditStrucHdl) &&
          !doIsWhiteSpaceAtOffset(selStart,textEditStrucHdl) &&
          !doIsWhiteSpaceAtOffset(selEnd - 1,textEditStrucHdl) &&
           doIsWhiteSpaceAtOffset(selEnd,textEditStrucHdl))
        {
          if(doGetCharAtOffset(selEnd,textEditStrucHdl) == ' ')
            (*textEditStrucHdl)->selEnd++;
        }

        // if insertion is after selected text, move insertion point back by size of selection

        if(gInsertPosition > selStart)
        {
          selEnd = (*textEditStrucHdl)->selEnd;
          gInsertPosition -= (selEnd - selStart);
          totalTextStart -= (selEnd - selStart);
        }

        // ... ... ... ...  ... ... ... ... ...
```

```
      additionalChars = doInsertTextAtOffset(gInsertPosition,textDataPtr,textSize,
                                             textEditStrucHdl);

      // ... ... ... ... ... if inserting multiple blocks of text, update insertion point for next block

      gInsertPosition += textSize + additionalChars;
      totalTextSize += textSize + additionalChars;

      // ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... dispose of
nonrelocatable block

      DisposePtr(textDataPtr);
    }
  }
  // ..................................... select total inserted text and adjust scrollbar

  TESetSelect(totalTextStart,totalTextStart + totalTextSize,textEditStrucHdl);
  doAdjustScrollbar(windowRef);

  // .................. set window's "touched" flag, and save post-insert selection start and end for Redo

  docStrucPtr->windowTouched = true;

  if(dragAttributes & kDragInsideSenderWindow)
  {
    docStrucPtr->postDropSelStart = totalTextStart;
    docStrucPtr->postDropSelEnd = totalTextStart + totalTextSize;
  }

  // ..................... if text edit structure had to be activated earlier, deactivate it

  if(!wasActive)
    TEDeactivate(textEditStrucHdl);

  return noErr;
}

// ************************************************************** doIsWhiteSpaceAtOffset

Boolean  doIsWhiteSpaceAtOffset(SInt16 offset,TEHandle textEditStrucHdl)
{
  char theChar;

  if((offset < 0) || (offset > (*textEditStrucHdl)->teLength - 1))
    return true;

  theChar = ((char *) *((*textEditStrucHdl)->hText))[offset];
```

```
  return (doIsWhiteSpace(theChar));
}

// ************************************************************************** doIsWhiteSpace

Boolean  doIsWhiteSpace(char theChar)
{
  return ((theChar == ' ') || (theChar == 0x0D));
}

// ************************************************************************** doGetCharAtOffset

char  doGetCharAtOffset(SInt16 offset,TEHandle textEditStrucHdl)
{
  if(offset < 0)
    return 0x0D;

  return (((char *) *((*textEditStrucHdl)->hText))[offset]);
}
// ************************************************************************** doInsertTextAtOffset

SInt16  doInsertTextAtOffset(SInt16 textOffset,Ptr textDataPtr,SInt32 textSize,
                             TEHandle textEditStrucHdl)
{
  SInt16 charactersAdded = 0;

  if(textSize == 0)
    return charactersAdded;

  // ...... if inserting at end of word, and selection does not begin with a space, insert a space

  if(!doIsWhiteSpaceAtOffset(textOffset - 1,textEditStrucHdl) &&
     doIsWhiteSpaceAtOffset(textOffset,textEditStrucHdl) &&
     !doIsWhiteSpace(textDataPtr[0]))
  {
    TESetSelect(textOffset,textOffset,textEditStrucHdl);
    TEKey(' ',textEditStrucHdl);
    ++textOffset;
    ++charactersAdded;
  }

  // ... if inserting at beginning of word and selection does not end with a space, insert space

  if(doIsWhiteSpaceAtOffset(textOffset - 1,textEditStrucHdl) &&
    !doIsWhiteSpaceAtOffset(textOffset,textEditStrucHdl) &&
    !doIsWhiteSpace(textDataPtr[textSize - 1]))
  {
    TESetSelect(textOffset,textOffset,textEditStrucHdl);
```

```
    TEKey(' ',textEditStrucHdl);
    ++charactersAdded;
  }

  // ..................................... before inserting, set selection range to a zero

  TESetSelect(textOffset,textOffset,textEditStrucHdl);
  TEInsert(textDataPtr,textSize,textEditStrucHdl);

  return charactersAdded;
}

// ************************************************************** doSavePreInsertionText

Boolean  doSavePreInsertionText(docStructurePointer  docStrucPtr)
{
  OSErr  osError;
  Size   tempSize;
  Handle tempTextHdl;

  if(docStrucPtr->preDragText == NULL)
    docStrucPtr->preDragText = NewHandle(0);
  tempTextHdl = (*(docStrucPtr->textEditStrucHdl))->hText;
  tempSize = GetHandleSize(tempTextHdl);
  SetHandleSize(docStrucPtr->preDragText,tempSize);
  osError = MemError();
  if(osError != noErr)
  {
    doErrorAlert(eDragUndo);
    return false;
  }

  BlockMove(*tempTextHdl,*(docStrucPtr->preDragText),tempSize);

  docStrucPtr->preDragSelStart = (*((docStrucPtr)->textEditStrucHdl))->selStart;
  docStrucPtr->preDragSelEnd   = (*((docStrucPtr)->textEditStrucHdl))->selEnd;

  return true;
}

// ************************************************************************** doUndoRedoDrag

void  doUndoRedoDrag(WindowRef windowRef)
{
  docStructurePointer docStrucPtr;
  Handle              tempTextHdl;
  Rect                portRect;
```

```
  docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));

  tempTextHdl = (*(docStrucPtr->textEditStrucHdl))->hText;
  (*(docStrucPtr->textEditStrucHdl))->hText = docStrucPtr->preDragText;
  docStrucPtr->preDragText = tempTextHdl;

  if(gUndoFlag)
  {
    (*((docStrucPtr)->textEditStrucHdl))->selStart = docStrucPtr->preDragSelStart;
    (*((docStrucPtr)->textEditStrucHdl))->selEnd = docStrucPtr->preDragSelEnd;
  }
  else
  {
    (*((docStrucPtr)->textEditStrucHdl))->selStart = docStrucPtr->postDropSelStart;
    (*((docStrucPtr)->textEditStrucHdl))->selEnd = docStrucPtr->postDropSelEnd;
  }

  gUndoFlag = !gUndoFlag;

  TECalText(docStrucPtr->textEditStrucHdl);
  GetWindowPortBounds(windowRef,&portRect);
  InvalWindowRect(windowRef,&portRect);
}

// ********************************************************************************
```

Demonstration Program Drag Comments

When this program is run, the user should open the included document "Drag Document" and drag selections to other locations within the document, to other demonstration program windows, to the windows of other applications that accept 'TEXT' format data, and to the desktop and Finder windows (to create text clippings). The user should also drag text from the windows of other applications to the demonstration program's windows.

The user should note the following:

- The highlighting of the demonstration program's windows, and of the proxy icon, when items containing data of the 'TEXT' flavor are dragged over those windows.
- The movement of the insertion point caret with the cursor when items are dragged within the demonstration program's windows, and the "hiding" of the caret when the drag originates in a demonstration program window and the cursor is moved over the selection.
- When dragging and dropping within the demonstration program's windows:
    - The program's implementation of "smart drag and drop." (For example, if there is a space character immediately to the left and right of the selection, the deletion is extended to include the second space character, thus leaving a single space character between the two words that previously bracketed the selection.)
    - The availability and effect of the Undo/Redo item in the Edit menu.

The non-drag and drop aspects of this program are based on the demonstration program `MonoTextEdit` (Chapter 21), and the contents of `Drag.h` and `Drag.c` are very similar to the contents of `MonoTextEdit.c`. Accordingly, comments on the content of `Drag.h` and `Drag.c` are restricted to those areas where modifications have been made to the code contained in `MonoTextEdit.c`.

Drag.h

Defines

Three additional constants are established for drag and drop errors.

typedefs

The docStructure data type has been extended to include fields to store the owning window's window reference, a Boolean which is set to true when the contents of the

window have been modified, and five fields to support drag-and-drop undo/redo.

Drag.c

Global Variables

`gDragTrackingHandlerUPP` and `gDragTrackingReceiverUPP` will be assigned universal procedure pointers to the tracking and receive handlers. `gEnableDragUndoRedoItem` and `gUndoFlag` will be used to control enabling/disabling of the Drag and Drop Undo/Redo item in the Edit menu.

main

Universal procedure pointers are created for the drag tracking and receive handlers (`dragTrackingHandler` and `dragReceiveHandler`).

windowEventHandler

When the `kWindowEventClickContentRgn` event type is received, `ConvertEventReftoEventRecord` is called and the address of the event record is passed in the call to `doInContent`, in addition to the mouse location and Shift key position. The address of an event record will be required by the call to `doStartDrag` in `doInContent`, and ultimately by the Drag Manager function TrackDrag.

doKeyEvent

The global variable `gEnableDragUndoRedoItem` is set to false. This causes the Drag and Drop Undo/Redo item in the Edit menu to be disabled.

doInContent

If the mouse-down was within the TextEdit view rectangle, `TEGetHiliteRgn` is called to attempt to get the highlight region. If there is a highlight region (that is, a selection), and if the mouse-down was within that region, `WaitMouseMoved` is called. `WaitMouseMoved` waits for either the mouse to move from the given initial mouse location or for the mouse button to be released. If the mouse moves away from the initial mouse location before the mouse button is released, `WaitMouseMoved` returns true, in which case the function `doStartDrag` is called.

doNewDocWindow

A nonrelocatable block is created for the window's document structure.

SetWindowProxyCreatorAndType is called with 0 passed in the fileCreator parameter and 'TEXT' passed in the fileType parameter to cause the system's default icon for a document file to be displayed as the proxy icon. In this program, the proxy icon is used solely for the purpose of demonstrating proxy icon highlighting when ShowDragHilite tis called to indicate that the window is a valid drag-and-drop target.

After gNumberOfWindows is incremented, four of the fields of the document structure are initialised.

Following the call to TEFeatureFlag, the drag tracking and receive handlers are installed on the window. (Note that the pointer to the window's document structure is passed in the handlerRefCon parameter of the installer functions.) If either installation is unsuccessful, the window and document structure are disposed of, the tracking handler also being removed in the case of a failure to install the receive handler.

doAdjustMenus

When the global variable gEnableDragUndoRedoItem is set to true, the Drag and Drop Undo/Redo item is enabled, otherwise it is disabled. If the item is enabled, the global variable gUndoFlag controls the item text, setting it to either Undo or Redo.

doEditMenu

If the Drag and Drop Undo/Redo item is chosen from the Edit menu, the function doUndoRedoDrag is called.

doAdjustCursor

After the first call to DiffRgn (which establishes the arrow and IBeam regions), a pointer to the window's document structure is retrieved. This allows the handle to the window's TextEdit structure to be passed in a call to TEGetHiliteRgn. The region returned by TEGetHiliteRgn is in local coordinates, so the next three lines change it to global coordinates preparatory to a call to DiffRgn. The DiffRgn call, in effect, cuts the equivalent of the highlight region out of the IBeam region.

If the location of the mouse (returned by the call to GetGlobalMouse) is within the highlight region, the cursor is set to the arrow shape.

doCloseWindow

If the preDragText field of the window's document structure does not contain NULL, DisposeHandle is called to release memory assigned in support of drag and drop redo/ undo.

The calls to `RemoveTrackingHandler` and `RemoveReceiveHandler` remove the tracking and receive handlers before the window is disposed of.

StartAndTrackDrag.c

doStartDrag

The call to NewDrag allocates a new drag object.

The call to `AddDragItemFlavor` creates a drag item and adds a data flavor (specifically 'TEXT') to that item. Note that the item reference number passed is 1. If additional flavors were to be added to the item, this same item reference number would be passed in the additional calls to `AddDragItemFlavor`.

The next block gets and sets the drag image for translucent dragging, but executes only if the program is running on Mac OS 8/9. `GetRegionBounds` gets the bounding rectangle of the highlight region and `OffsetRect` adjusts the coordinates so that top left is 0,0. The call to `NewGWorld` creates an 8-bit deep offscreen graphics world the same size as this rectangle. `CopyBits` is then called to copy the highlight region area to the offscreen graphics world. The calls to `CopyRgn` and `OffsetRgn` create a region the same shape as the highlight region and adjusted so that the top left of the bounding rectangle is 0,0. The next two lines establish the offset point required by the following call to `SetDragImage`. This offset is required to move the pixel map in the offscreen graphics world to the global coordinates where the drag image is to initially appear. Finally, the handles to the offscreen graphics world and mask, the offset, and a constant specifying the required level of translucency are passed in the call to `SetDragImage`, which associates the image with the drag reference.

Any errors that might occur in setting up the translucent drag are not critical on Mac OS 8/9 because the next block creates the drag region for the alternative visual representation (a gray outline). On Mac OS X, this is the block that creates the gray outline. The received highlight region (which is in local coordinates) is copied to `dragRgnHdl`, which is then converted to global coordinates. This region, in turn, is copied to `tempRgnHdl`, which is then inset by one pixel. The call to `DiffRgn` subtracts the inset region from `dragRgnHdl`, leaving the latter with the same outline as the highlight region but only one pixel thick. The newly defined drag region is passed in the theRegion parameter of the call to `TrackDrag`.

`TrackDrag` performs the drag. During the drag, the Drag Manager follows the cursor on the screen with the translucent image (Mac OS 8/9) or gray outline (Mac OS X) and sends tracking messages to applications that have registered drag tracking handlers. When the user releases the mouse button, the Drag Manager calls any receive drop handlers that have been registered on the destination window.

The `TrackDrag` function returns `noErr` in situations where the user selected a destination for the drag and the destination received data from the Drag Manager. If the user drops over a non-aware application or the receiver does not accept any data from the Drag Manager, the Drag Manager automatically provides a "zoom back" animation and returns `userCanceledErr`. Thus the first return will execute only if an error other than userCanceledErr was returned.

dragTrackingHandler

`dragTrackingHandler` is the drag tracking handler. It is a callback function.

Firstly, if the message received is not the enter handler message, and if it was determined at the time of receipt of the enter handler message that the drop cannot be accepted, the function returns immediately.

The pointer received in the `handlerRefCon` formal parameter is cast to a pointer to the window's document structure so that certain fields in this structure can later be accessed.

`GetDragAttributes` gets the current set of drag attribute flags. `GetCaretTime` gets the insertion point caret blink interval. `TickCount` gets the current system tick count. These latter two values will be used by the caret drawing code.

Enter Handler

Within the switch, the "enter handler" message is processed at the first case.

`CountDragItems` returns the number of items in the drag. Then, for each of the items in the drag, `GetDragItemReferenceNumber` is called to retrieve the item reference number, allowing the call to `GetFlavorFlags` to determine whether the item contains data of the 'TEXT' flavor. (`GetFlavorFlags` will return an error if the flavor does not exist.) `gCanAcceptItems` is assigned false if any item does not contain 'TEXT' data, meaning that the drag will only be accepted if all items contain data of the 'TEXT' flavor.

Enter Window

If the message is the "enter window" message, several global variables are initialized. These globals will be used when the "in window" message is received to control insertion point caret drawing and window highlighting.

In Window

Each time the "in window" message is received, `GetDragMouse` is called to get the current mouse location in global coordinates. The location is copied to a local Point

variable, which is then converted from global to local coordinates.

The first if block executes if the drag has left the sender window. If the mouse cursor is within the window's TextEdit view rectangle, and if `gCursorInContent` has previously been set to false (recall that it is set to false at the "enter window" message), `ShowDragHilite` is called to draw the standard drag and drop highlight around the view rectangle. `gCursorInContent` is then set to true to defeat further `ShowDragHilite` calls while the drag remains in this window.

If the mouse cursor is not within the window's TextEdit view rectangle, and if `gCursorContent` has previously been set to true (causing a highlight draw), `HideDragHilite` is called to remove the highlighting, and `gCursorInContent` is set to false again so that `ShowDragHilite` is called if the drag moves back inside a view rectangle again.

With window highlighting attended to, the next task is to perform insertion point caret drawing.

The function `doGetOffset` takes the mouse location in global coordinates and the window's TextEdit structure handle and returns the offset into the text corresponding to the mouse location. (Note that doGetOffset will return -1 if the cursor is not within the content region and the view rectangle of the window under the cursor. Note also that, if the cursor is within the content region of the window under the cursor, `doGetOffset` sets that window's graphics port as the current port.)

If the drag is currently inside the sender window, and the offset returned by `doGetOffset` indicates that the cursor is within the TextEdit selection, theOffset is set to -1. As will be seen, this defeats the drawing of the caret when the cursor is within the selection.

The offset returned by `doGetOffset` is then saved to a global variable. As will be seen, the value assigned to this variable the last time the "in window" case executes (when the user releases the mouse button) will be used by the receive handler `dragReceiveHandler`.

If the offset has changed since the last time the "in window" case executed, the caret flashing timer is reset to the time the handler was entered this time around and the caret show/hide flag is set to "show." The current offset is then assigned to the global `gLastOffset` preparatory to the execution of the "in window" case.

If the caret flashing interval has elapsed, the show/hide flag is toggled and the caret flashing timer is reset.

If the caret show/hide flag indicates that the caret should be "hidden," theOffset will be set

to -1 to defeat caret drawing.

If the offset has changed since the last execution of the "in window" case, the function for drawing/erasing the caret is called in certain circumstances. Firstly, if this is not the first execution of the "in window" case since entering the window, `doDrawCaret` is called to erase the caret previously drawn at the old offset. Secondly, if the show/hide flag indicates that the caret should be drawn, `doDrawCaret` is called to draw the caret at the current offset.

The global that stores the old offset is assigned the current offset before the case exits.

Leave Window

When the "leave window" message is received, if the caret is on the screen, it is erased. If the window highlighting has previously been drawn, `HideDragHilite` is called to remove the highlighting.

doGetOffset

`doGetOffset` is called by `dragTrackingHandler` to return the offset into the text corresponding to the specified mouse location. `doGetOffset` also sets the graphics port to the port associated with the window under the cursor.

If the part code returned by FindWindow indicates that the cursor is within the content region of a window, that window's graphics port is set as the current port and the cursor location is converted to local coordinates preparatory to the call to `PtInRect`.

If the cursor is within the view rectangle of the `TextEdit` structure associated with the window, `TEGetOffset` is called to get the offset into the text corresponding to the cursor location and `TEGetPoint` is called to get the local coordinates of the character immediately before the offset.

If the offset is at a TextEdit line start, and the character immediately before the offset is not a carriage return, and the horizontal coordinate of the character immediately before the offset is less than the horizontal coordinate of the cursor, `theOffset` is decremented by one. In the situation where the cursor is dragged to the right of the rightmost character of a line, this will cause the caret to continue to be drawn immediately to the right of that character instead of "jumping" to the beginning of the next line.

isOffsetAtLineStart

`isOffsetAtLineStart` is called by `doGetOffset`. It returns true if the specified offset is at a TextEdit line start.

doDrawCaret

doDrawCaret is called by dragTrackingHandler to draw and erase the caret. The transfer mode is set to patXor so that two successive calls will, in effect, draw and erase the image.

The call to TEGetPoint gets the coordinates of the bottom left of the character at the specified offset. The call to the function doGetLine returns the TextEdit line number that contains the specified offset.

The next block accommodates a quirk of TextEdit. For some reason, TextEdit does not return the proper coordinates of the last offset in the field if the last character in the record is a carriage return. TEGetPoint returns a point that is one line higher than expected. This block fixes the problem.

Following the call to PenMode, TEGetHeight is called to get the height of a single line of text. A line is then drawn from a point one pixel to the left and above the bottom left of the character at the offset to a point vertically above the starting point. The length of the line is one pixel less that the line height.

doGetLine

doGetLine is called by doDrawCaret. It returns the TextEdit line number that contains the specified offset.

ReceiveAndUndoDrag.c

dragReceiveHandler

dragReceiveHandler is the drag receive handler. It is a callback function.

Recall that dragTrackingHandler sets gCanAcceptItems to true if all items in the drag contained data of the 'TEXT' flavor. Recall also that dragTrackingHandler sets gInsertPosition to -1 if the cursor is over the selection. Thus, if at least one item does not contain data of the 'TEXT' flavor, or if the cursor is over the selection at the time the mouse button is released, the function exits and dragNotAcceptedErr is returned to the Drag Manager. dragNotAcceptedErr causes the Drag Manager to execute a "zoomback" animation of the drag region to the source location.

The pointer received in the handlerRefCon formal parameter is cast to a pointer to the window's document structure. The handle to the TextEdit structure associated with the window is then retrieved from the document structure.

`SetPortWindowPort` sets the window's graphics port as the current port. The next line saves whether the TextEdit structure is currently active or inactive so that the state can later be restored. If the TextEdit structure is currently not active, it is made active.

`GetDragAttributes` and `GetDragModifiers` are called to get the drag's attributes and the modifier keys that were pressed at mouse-down and mouse-up time.

If there are multiple items in the drag, the initial insertion point, which is set in the drag tracking handler, is saved to a local variable for later use by `TESetSelect`. (If there are multiple items to insert, the offset in `gInsertPosition` will be updated each time the main if block executes.)

`CountDragItems` returns the number of items in the drag and the main for loop executes for each item. Within the loop, the first call (to `GetDragItemReferenceNumber`) retrieves the item's item reference number, which is passed in the call to `GetFlavorDataSize` to get the size of the 'TEXT' data. If `GetFlavorDataSize` does not return an error, the following occurs:

- The data size returned by `GetFlavorDataSize` is added to the current size of the text in the `TextEdit` structure. If adding the 'TEXT' data to the current text would exceed the `TextEdit` limit, the handler exits, returning `dragNotAcceptedErr` to the Drag Manager.
- A nonrelocatable block the size of the 'TEXT' data is created and `GetFlavorData` is called to get the 'TEXT' data into that block.
- If the insertion point caret is on the screen, it is removed. (Recall that the global variables used here are set in the drag tracking handler.) If the window is currently highlighted, the highlighting is removed.
- If the drop is within the sender window, the function `doSavePreInsertionText` is called to save the current `TextEdit` text and the current selection start and end. This is to support drag undo/redo. If `doSavePreInsertionText` returns true, flags are set to cause the Undo/Redo item in the Edit menu to be enabled and to cause the initial item text to be set to Undo.
- If there are multiple items in the drag, the initial insertion point is saved for later use by `TESetSelect`. If there are multiple items to insert, the offset in `gInsertPosition` will be updated each time around.
- The variable `moveText` is assigned true if the drop is inside the sender window and the option key was not down when the mouse button was pressed or released. If `moveText` is true, the current selection must be deleted, so the if block executes.
    - Firstly, the current selection start and end are saved to two local variables.
    - The next block implements "smart drag and drop." If the character just before selection start is a space or CR character, and if the first character in the selection is not such a character, and if the last character in the selection is not such a character, and if the character just after the selection is such a

character, then, if the character just after the selection is a space character, the current end of the selection is extended to include that space character. This means that if, for example, just the characters of a word are currently selected, the space character immediately after the selection is added to the selection so that only one space character will remain between the words which bracket the selection when the selection is deleted by `TEDelete`.
    - If the current drop insertion offset is after the selection start offset, the local variable holding the selection end offset is updated (it may have been increased by the "smart drag and drop" code), and `gInsertPosition` and `totalTextStart` offsets are moved back by the length of the selection.
    - `TEDelete` then deletes the selected text (perhaps extended by one by the "smart drag and drop" code) from the TextEdit structure and redraws the text.
- The `doInsertTextAtOffset` function is called to insert the 'TEXT' data at the current insertion point. This function implements another aspect of "smart drag and drop," which can possibly add additional characters to the `TextEdit` structure. The number of additional characters added (if any) is returned by the function.
- If the main if block executes again (meaning that there are multiple items in the drag), `gInsertPosition` must be updated to the offset for the next insertion. This is achieved by adding the size of the last insertion, plus the number of any additional characters added by do`InsertTextAtOffset` to the current value in `gInsertPosition`. The value in `totalTextSize` is increased by the same amount.

When all items have been inserted, the main if block exits. `TESetSelect` is then called to set the selection range to the total inserted text. This call unhighlights the previous selection and highlights the new selection range. `doAdjustScrollbar` is called to adjust the scroll bars.

The `windowTouched` field in the document structure is set to `true` to record the fact that the contents of the window have been modified. The post-insertion selection start and selection end offsets are saved to the relevant fields of the document structure for possible use in the application's Undo/Redo function.

Finally, if the TextEdit structure was not active when `dragReceiveHandler` was entered, it is restored to that state.

isWhiteSpaceAtOffset

`isWhiteSpaceAtOffset` is called by `dragReceiveHandler` and `doInsertTextAtOffset`. Given an offset into a TextEdit structure, it determines if the character at that offset is a white space character, that is, a space character or a carriage return character.

isWhiteSpace

`isWhiteSpace` is called by `isWhiteSpaceAtOffset` and `doInsertTextAtOffset`. Given a character, it returns true if that character is either a space character or a carriage return character.

doGetCharAtOffset

`doGetCharAtOffset` is called by `dragReceiveHandler`. Given an offset and a handle to a `TextEdit` structure, it returns the character at that offset.

doInsertTextAtOffset

`doInsertTextAtOffset` is called by `dragReceiveHandler` to insert the drag text at the specified offset. In addition to inserting the text, `dragReceiveHandler` implements additional aspects of "smart drag and drop," returning the number of space characters, if any, added to the TextEdit structure text by this process.

If there is no text in the buffer, the function simply returns.

If the character to the left of the insertion offset is not a space, and if the character at the offset is a space, the insertion is at the end of a word. If, in addition, the first character in the drag is not a space, `TESetSelect` is called to collapse the selection range to an insertion point at the received offset, `TEKey` is called to insert a space character in the `TextEdit` structure at that offset, the offset is incremented by one to accommodate that added character, and the variable that keeps track of the number of added characters is incremented.

The next block is similar, except that it inserts a space character if the insertion is at the beginning of a word and the text to be inserted does not end with a space character. Also, in this block, the offset is not incremented.

With the "smart drag and drop" segment completed, `TESetSelect` is called to ensure that the selection range is collapsed to an insertion point at the (possibly incremented) offset, and `TEInsert` is called to insert the text into the `TextEdit` structure at that insertion point.

doSavePreInsertionText

`doSavePreInsertionText` is called by `dragReceiveHandler` to save the document's pre-drop text and selection in support of drag and drop undo/redo.

If the `preDragText` field of the window's document structure currently contains `NULL`, a new empty relocatable block is created and its handle assigned to the `preDragText` field.

The next block copies the handle to the `TextEdit` structure's text to a local variable, gets the size of the `TextEdit` structure's text, and expands the newly created block to that size.

`BlockMove` is then called to copy the `TextEdit` structure's text to the newly created block. In addition, the current (pre-drop) selection start and end are saved to the window's document structure.

doUndoRedoDrag

`doUndoRedoDrag` is called by `doEditMenu` in the event of the user choosing the Drag and Drop Undo/Redo item in the Edit menu. That item will only be enabled when the user has released the mouse button and the drop is within the sender window.

The first block means that, each time this function is called, the text block whose handle is assigned to the TextEdit structure will toggle between the pre-drop text and the post-drop text. The first time this function is called, the item text in the Edit Menu will be "Drag and Drop Undo" and the TextEdit structure will be assigned the handle to pre-drop text saved in the document structure's preDragText field. The next time the function is called, the item text in the Edit menu will be "Drag and Drop Redo" and the `TextEdit` structure will be assigned the handle to post-drop text, and so on.

The global variable `gUndoFlag` is toggled by this function. At the next block, and depending on the value in `gUndoFlag`, either the pre-drop or post-drop selection start and end offsets are assigned to the TextEdit structure, as appropriate. (`gUndoFlag` also controls the text in the Drag and Drop Undo/Redo item in the Edit menu. See doAdjustMenus.)

With the appropriate text and selection assigned to the window's `TextEdit` structure, `TECalText` is called to wrap the text to the width of the view rectangle and re-calculate the line-starts. Finally, `InvalWindowRect` is called to force a call to `TEUpdate` (in the doDrawContent function) to redraw the text.

Chapter 24. Basic Sound and Speech

IN THIS CHAPTER

- Introduction to Sound
- Sound Production
- Sound Recording-Mac OS 8/9
- Speech
- Relevant Constants, Data Types, and Functions
- The AsynchSoundLib Library
- Demonstration Program SoundAndSpeech Listing
- Demonstration Program SoundAndSpeech Comments

Introduction to Sound

On the Macintosh, the hardware and software aspects of producing and recording sounds are tightly integrated.

Audio Hardware and Sound-Related System Software

The audio hardware includes a speaker or speakers, a microphone, and one or more integrated circuits that convert digital data to analog signals and vice versa.

The sound-related system software managers are as follows:

- **The Sound Manager.** The Sound Manager provides the ability to
    - Play sounds through the speakers.
    - Manipulate sounds, that is, vary such characteristics as loudness, pitch, timbre, and duration.
    - Compress sounds so that they occupy less disk space.
- **The Sound Input Manager.** The Sound Input Manager provides the ability to record sounds using a microphone or other sound input device.
- **The Speech Manager.** The Speech Manager provides the ability to convert text into spoken words.

Sound Input and Output Capabilities

The basic audio hardware, together with the sound-related system software, provides for the following capabilities:

- The playing back of digitally recorded sounds. (Digitally recorded sound is referred to as **sampled** sound.
- The playing back of simple sequences of notes or of complex waveforms.
- The recording of sampled sounds.
- The conversion of text to spoken words.
- The mixing and synchronization of multiple channels of sampled sounds.
- The compression and decompression of sound data.
- The integration and synchronization of sound production with the display of video and still images. (For example, the Sound Manager is used by QuickTime to handle all the sound data in a QuickTime movie.)

Basic and Enhanced Sound Capabilities

Users can enhance sound playback and recording quality by substituting better speakers and microphones. Audio capabilities may be further enhanced by adding an expansion card containing very high quality digital signal processing (DSP) circuitry, together with sound input or output hardware. Another enhancement option is to add a MIDI interface to one of the serial ports. Fig 1 illustrates the basic sound capabilities of the Macintosh and how those capabilities may be further enhanced and extended.

Sound Data

The Sound Manager can play sounds defined using one of the following kinds of sound data:

- **Square Wave Data.** Square-wave data can be used to play a simple sequence of sounds in which each sound is described by frequency (pitch), amplitude (volume), duration.

- **Wave-Table Data.** Wave table data may be used to produce more complex sounds than are possible using square-wave data. A wave cycle is represented as an array of bytes that describe the timbre (tone) of a sound at a point in the cycle.

- **Sampled-Sound Data.** Sampled sound data is a continuous list of relative voltages over time that allow the Sound Manager to reconstruct an arbitrary analog wave form. They are typically used to play back prerecorded sounds such as speech or special sound effects.

This chapter is oriented primarily towards the recording and playback of sampled sounds.

Sampled Sound

Two basic characteristics affect the quality of sampled sound. Those characteristics are sample rate and sample size.

Sample Rate

Sample rate, or the rate at which voltage samples are taken, determines the highest possible frequency that can be recorded. Specifically, for a given sample rate, sounds can be sampled up to half that frequency. For example, if the sample rate is 22,254 samples per second (that is, 22,254 hertz, or Hz), the highest frequency that can be recorded is about 11,000 Hz. A commercial compact disc is sampled at 44,100 Hz, providing a frequency response of up to about 20,000 Hz, which is the limit of human hearing.

Sample Size

Sample size, or quantisation, determines the dynamic range of the recording (the difference between the quietest and the loudest sound). If the sample size is eight bits, 256 discrete voltage levels can be recorded. This provides approximately 48 decibels (dB) of dynamic range.

A compact disc's sample size is 16 bits, which provides about 96 dB of dynamic range. (Humans with good hearing are sensitive to ranges greater than 100 dB.)

Sound Manager Capabilities

The Sound Manager supports 16-bit stereo audio samples with sample rates up to 64kHz.

Storing Sampled Sounds

Sampled-sound data comprises a series of sample frames. You can use the Sound Manager to store sampled sounds in one of two ways, either in sound resources or in sound files.

Sound Components

The Sound Manager uses **sound components** to modify sound data. A sound component is a stand-alone code resource that can perform operations on sound data such as compression, decompression, and converting sample rates. Sound components may be hooked together in series to perform complex tasks, as shown in the example in Fig 2.

The Sound Manager is aware of the sound output device selected by the user and assembles a component chain suitable for producing the desired quality of sound on that device. Thus your application is generally unaware of the sound component chain assembled to produce a sound on the selected output device.

Compression/Decompression Components

Components that compress and decompress sound are called **codecs** (compression/ decompression components). Apple Computer supplies codecs that can handle 3:1 and 6:1 compression and expansion, which are suitable for most audio requirements. The Sound Manager can use any available codec to handle compression and expansion of audio data.

> **Note**
>
> A term closely associated with the subject of codecs is **MACE** (**Macintosh Audio Compression and Expansion**). MACE is a collection of Sound Manager functions that provide audio data compression and expansion capabilities in ratios of either 3:1 or 6:1. The Sound Manager uses codecs to handle the MACE capabilities.

Sound Resources

A sound resource is a resource of type `'snd '` that contains sound commands and possibly also sound data. Sound resources provide a simple way for you to incorporate sounds into your application.

Sound Production

Sound Channels

A Macintosh produces sound when the Sound Manager sends data through a sound channel to the audio hardware. A sound channel is basically a queue of sound commands (see the following), which might be placed into the sound channel by your application or by the Sound Manager itself.

The Sound Manager uses the `SndChannel` data type to define a sound channel:

```
struct SndChannel
{
  SndChannelPtr  nextChan;       // Pointer to next channel.
  Ptr            firstMod;       // (Used internally.)
  SndCallBackUPP callBack;       // Pointer to callback function.
  long           userInfo;       // Free for application's use.
  long           wait;           // (Used internally.)
  SndCommand     cmdInProgress;  // (Used internally.)
  short          flags;          // (Used internally.)
  short          qLength;        // (Used internally.)
  short          qHead;          // (Used internally.)
  short          qTail;          // (Used internally.)
  SndCommand     queue[128];     // (Used internally.)
}
typedef struct SndChannel SndChannel;
typedef SndChannel *SndChannelPtr;
```

Multiple Sound Channels

It is possible to have several channels of sound open at one time. The Sound Manager (using the Apple Mixer sound component) mixes together the data coming from all open sound channels and sends a single stream of sound data to the current sound output device. This allows a single application to play two or more sounds at once. It also allows multiple applications to play sounds at the same time.

Sound Commands

When you call the appropriate Sound Manager function to play a sound, the Sound Manager issues one or more sound commands to the audio hardware. A sound command is an instruction to produce or modify sound, or otherwise contribute to the overall sound production process. The structure of a sound command is defined by the `SndCommand` data type:

```
struct SndCommand
{
  unsigned short cmd;     // Command number.
  short          param1;  // First parameter.
  long           param2;  // Second parameter.
} ;
typedef struct SndCommand SndCommand;
```

The Sound Manager provides a rich set of sound commands, which are defined by constants. Some examples are as follows:

```
quietCmd  = 3   Stop the sound currently playing.
flushCmd  = 4   Remove all commands currently queued in specified sound channel.
syncCmd   = 14  Synchronise multiple channels of sound.
soundCmd  = 80  Install a sampled sound as a voice in a channel.
bufferCmd = 81  Play a buffer of sampled-sound data.
```

Note

Several Sound Manager sound commands are not available in Carbon.

Sound Commands In 'snd ' Resources

A simple way to issue sound commands is to call the function `SndPlay`, specifying a sound resource of type `'snd'` that contains the sound commands you want to issue.

Often, a `'snd '` resource consists only of a single sound command (usually the `bufferCmd` command) together with data that describes a sampled sound to be played. The following is an example of such a `'snd '` resource, shown in the form of the output of the MPW tool DeRez when applied to the resource:

```
data 'snd ' (19068,"My sound",purgeable)
{
  /* Sound resource header */
    $"0001"      /* Format type. */
    $"0001"      /* Number of data types. */
    $"0005"      /* Sampled-sound data. */
    $"00000080"  /* Initialisation option: initMono. */
  /* Sound commands */
    $"0001"      /* Number of sound commands that follow (1). */
    $"8051"      /* Command 1 (bufferCmd). */
    $"0000"      /* param1 = 0. */
    $"00000014"  /* param2 = offset to sound header (20 bytes). */
  /* Sampled sound header (Standard sound header)*/
    $"00000000"  /* samplePtr      Pointer to data (it follows immediately). */
    $"00000BB8"  /* length         Number of bytes in sample (3000 bytes). */
    $"56EE8BA3"  /* sampleRate     Sampling rate of this sound (22 kHz). */
    $"000007D0"  /* loopStart      Starting of the sample's loop point. */
    $"00000898"  /* loopEnd        Ending of the sample's loop point. */
    $"00"        /* encode         Standard sample encoding. */
    $"3C"        /* baseFrequency  BaseFrequency at which sample was taken. */
      /* sampleArea[]  Sampled sound data */
    $"80 80 81 81 81 81 81 81 80 80 80 80 80 81 82 82"
    $"82 83 82 82 81 80 80 7F 7F 7F 7E 7D 7D 7D 7C 7C"
    (Rest of sampled sound data.)
} ;
```

Note that the sound resource header section indicates that the sound is defined using sampled-sound data. Note also that the sound commands section contains a call to a single sound command (the `bufferCmd` command (`0x51`)) and that the offset bit of the command number is set to indicate that the sound data is contained within the resource itself. (Data can also be stored in a buffer separate from a sound resource.) The second parameter to the `bufferCmd` command indicates the offset from the beginning of the resource to the sampled sound header, which immediately follows the sound commands section.

Note

The sampled sound header shown is a standard sound header, which can reference only buffers of monophonic 8-bit sound. The extended sound header is used for 8-bit or 16-bit stereo sound data as well as monophonic sound data. The compressed sound header is used to describe compressed sound data, whether monophonic or stereo.

Note that the first part of the sampled sound header contains information about the sample and that the sampled sound data is part of the sampled sound header.

Sending Sound Commands Directly from the Application

You can also send sound commands into a sound channel one at a time by calling `SndDoCommand` or you can bypass a sound queue altogether by calling the `SndDoImmediate`.

Synchronous and Asynchronous Sound

You can play sounds either synchronously or asynchronously. When your application plays a sound synchronously, it cannot continue executing until the sound has finished playing. When your application plays a sound asynchronously, it can continue other processing while the sound is playing.

From a programming standpoint, asynchronous sound production is considerably more complex than synchronous sound production.

Playing a Sound

Note

The Sound Manager function `SndStartFilePlay` (starts a file playing from disk), together with the associated functions `SndPauseFilePlay`, `SndStopFilePlay`, `SndPlayDoubleBuffer`, are not available in Carbon.

Playing a Sound Resource

You can load a sound resource into memory and then play it using the `SndPlay` function. As previously stated, a `'snd '` resource contains sound commands that play the desired sound and might also contain sound data. If the sound data is compressed, `SndPlay` decompresses the data in order to play the sound.

Channel Allocation

If you pass `NULL` in the first parameter of `SndPlay`, a sound channel will be automatically allocated to play the sound and then automatically disposed of when the sound has finished playing.

Playing Sounds Asynchronously

The Sound Manager allows you to play sounds asynchronously only if you allocate sound channels yourself. If you use such a technique, your application will need to dispose of a sound channel whenever the application finishes playing a sound. In addition, your application might need to release a sound resource that you played on a sound channel.

The Sound Manager provides certain mechanisms that allow your application to ascertain when a sound finishes playing, so that it can arrange to dispose of a sound channel no longer being used and other data (such as a sound

resource) that you no longer need after disposing of the channel. Despite the existence of these mechanisms, the programming aspects of asynchronous sound remain rather complex. For that reason, the demonstration program files associated with this chapter include a library, called `AsynchSoundLib`, which support asynchronous sound playback and which eliminates the necessity for your application to itself include source code relating to the more complex aspects of asynchronous sound management.

`AsynchSoundLib`, which may be used by any application that requires a straightforward and uncomplicated interface for asynchronous sound playback, is documented following the Constants, Data Types, and Functions section of this chapter.

Sound Recording-Mac OS 8/9

On Mac OS 8/9, the Sound Input Manager provides a high-level function that allows your application to record sounds from the user and store them in memory. When you call this function, the Sound Input Manager presents the sound recording dialog shown in Fig 3.

> **Note**
>
> The Sound Manager function `SndRecordToFile` (records sound data to a file) is not available in Carbon.

Recording a Sound Resource

You can record sounds from the current input device using the `SndRecord` function. When calling `SndRecord`, you can pass a handle to a block of memory in the fourth parameter. The incoming data will then be stored in that block, the size of which determines the recording time available. If you pass `NULL` in the fourth parameter, the Sound Input Manager allocates the largest possible block in the application heap. Either way, the Sound Input Manager resizes the block when the user clicks the Save button.

When you have recorded a sound, you can play it back by calling `SndPlay` and passing it the handle to the block of memory in which the sound data is stored. That block has the structure of a `'snd '` resource, but its handle is not a handle to an existing resource. To save the recorded data as a resource, you can use the appropriate Resource Manager functions in the usual way.

Recording Quality

One of the following constants should be passed in the third parameter of the `SndRecord` call so as to specify the recording quality required:

Constant	*Value*	*Meaning*
`siCDQuality`	`'cd '`	44.1kHz, stereo, 16 bit.
`siBestQuality`	`'best'`	22kHz, mono, 8 bit.
`siBetterQuality`	`'betr'`	22kHz, mono, 3:1 compression.

`siGoodQuality`	`'good'`	22KHz, mono, 6:1 compression

The highest quality sound naturally requires the greatest storage space. Accordingly, be aware that, for most voice recording, you should specify `siGoodQuality`.

As an example of the storage space required for sounds, one minute of monophonic sound recorded with the same fidelity as a commercial compact disc occupies about 5.3 MB of disk space, and one minute of telephone-quality speech takes up more than half a megabyte.

Speech

The Speech Manager converts text into sound data and passes to the Sound Manager. The Speech Manager utilises a speech synthesizer, which can include one or more voices, each of which may have different tonal qualities.

Generating Speech From a String

The `SpeakString` function is used to convert a text string into speech. `SpeakString` automatically allocates a speech channel, produces the speech on that channel, and then disposes of the speech channel.

Asynchronous Speech

Speech generation is asynchronous, that is, control returns to your application before `SpeakString` finishes speaking the string. However, you are free to release the memory allocated for the string as soon as `SpeakString` returns, the reason being that `SpeakString` copies the string into an internal buffer,.

Synchronous Speech

If you wish to generate speech synchronously, you can use `SpeakString` in conjunction with the `SpeechBusy` function, which returns the number of active speech channels, including the speech channel created by the `SpeakString` function.

Relevant Constants, Data Types, and Functions

Constants

Recording Qualities

```
siCDQuality     = FOUR_CHAR_CODE('cd  ')  44.1kHz, stereo, 16 bit.
siBestQuality   = FOUR_CHAR_CODE('best')  22kHz, mono, 8 bit.
siBetterQuality = FOUR_CHAR_CODE('betr')  22kHz, mono, MACE 3:1.
siGoodQuality   = FOUR_CHAR_CODE('good')  22kHz, mono, MACE 6:1.
```

Typical Sound Commands

```
quietCmd        = 3   Stop the sound currently playing.
flushCmd        = 4   Remove all commands currently queued in specified sound channel.
syncCmd         = 14  Synchronise multiple channels of sound.
soundCmd        = 80  Install a sampled sound as a voice in a channel.
bufferCmd       = 81  Play a buffer of sampled-sound data.
```

Data Types

Sound Channel Structure

```
struct SndChannel
{
  SndChannelPtr  nextChan;       // Pointer to next channel.
  Ptr            firstMod;       // (Used internally.)
  SndCallBackUPP callBack;       // Pointer to callback function.
  long           userInfo;       // Free for application's use.
  long           wait;           // (Used internally.)
  SndCommand     cmdInProgress;  // (Used internally.)
  short          flags;          // (Used internally.)
  short          qLength;        // (Used internally.)
  short          qHead;          // (Used internally.)
  short          qTail;          // (Used internally.)
  SndCommand     queue[128];     // (Used internally.)
}
typedef struct SndChannel SndChannel;
typedef SndChannel *SndChannelPtr;
```

Sound Command Structure

```
struct SndCommand
{
  unsigned short cmd;     // Command number.
  short          param1;  // First parameter.
  long           param2;  // Second parameter.
} ;
typedef struct SndCommand SndCommand;
```

Functions

Playing Sound Resources

```
void   SysBeep(short duration);
OSErr  SndPlay(SndChannelPtr chan,SndListHandle sndHdl,Boolean async);
```

Allocating and Releasing Sound Channels

```
OSErr  SndNewChannel(SndChannelPtr *chan,short synth,long init,SndCallBackUPP userRoutine);
OSErr  SndDisposeChannel(SndChannelPtr chan,Boolean quietNow);
```

Sending Commands to a Sound Channel

```
OSErr  SndDoCommand(SndChannelPtr chan,const SndCommand *cmd,Boolean noWait);
OSErr  SndDoImmediate(SndChannelPtr chan,const SndCommand *cmd);
```

Recording Sounds

```
OSErr  SndRecord(ModalFilterUPP filterProc,Point corner,OSType quality,
       SndListHandle *sndHandle);
```

Generating Speech

```
OSErr  SpeakString(ConstStr255Param textToBeSpoken);
short  SpeechBusy(void);
```

The AsynchSoundLib Library

The AsynchSoundLib library is intended to provide a straightforward and uncomplicated interface for asynchronous sound playback.

AsynchSoundLib requires that you include a global "attention" flag in your application. At startup, your application must call AsynchSoundLib's initialization function and provide the address of this attention flag. Thereafter, the application must continually check the attention flag within its main event loop.

AsynchSoundLib's main function is to spawn asynchronous sound tasks, and communication between your application and AsynchSoundLib is carried out on an as-required basis. The basic phases of communication for a typical sound playback sequence are as follows:

- Your application tells AsynchSoundLib to play some sound.
- AsynchSoundLib uses the Sound Manager to allocate a sound channel and begins asynchronous playback of your sound.
- The application continues executing, with the sound playing asynchronously in the background.
- The sound completes playback. AsynchSoundLib has set up a sound command that causes it (AsynchSoundLib) to be informed immediately upon completion of playback. When playback ceases, AsynchSoundLib sets the application's global attention flag.
- The next time through your application's event loop, the application notices that the attention flag is set and calls AsynchSoundLib to free up the sound channel.

When your application terminates, it must call AsynchSoundLib to stop any asynchronous playback in progress at the time.

AsynchSoundLib's method of communication with the application minimizes processing overhead. By using the attention flag scheme, your application calls AsynchSoundLib's cleanup function only when it is really necessary.

AsynchSoundLib Functions

The following documents those AsynchSoundLib functions that may be called from an application.

To facilitate an understanding of the following, it is necessary to be aware that AsynchSoundLib associates a data structure, referred to in the following as an ASStructure, with each channel. Each ASStructure includes the following fields:

```
SndChannel  channel;      // The sound channel.
SInt32      refNum;       // Reference number.
Handle      sound;        // The sound.
char        handleState;  // State to which to restore the sound handle.
Boolean     inUse;        // Is this ASStructure currently in use?

OSErr  AS_Initialise(attnFlag,numChannels);

Boolean *attnFlag;    Pointer to application's "attention" flag global variable.
SInt16  numChannels;  Number of channels required to be open simultaneously. If 0 is
                      specified, numChannels defaults to 4.

Returns:  0  No errors.
          Non-zero results of MemError call.
```

This function stores the address of the application's "attention" flag global variable and then allocates memory for a number of ASStructures equal to the requested number of sound channels.

```
OSErr  AS_PlayID(resID,refNum);

SInt16 resID    Resource ID of the 'snd ' resource.
SInt32 *refNum  A pointer to a reference number storage variable. Optional.

Returns:  0  No errors.
          1  No channels available.
          Non-zero results of ResError call.
          Non-zero results of SndNewChannel call.
          Non-zero results of SndPlay call.
```

This function initiates asynchronous playback of the `'snd'` resource with ID `resID`.

If you pass a pointer to a variable in their `refNum` parameters, `AS_PlayID` and its sister function `AS_PlayHandle` (see below) return a reference number in that parameter. As will be seen, this reference number may be used to gain more control over the playback process. However, if you simply want to trigger a sound and let it run to completion, with no further control over the playback process, you can pass `NULL` in the `refNum` parameter. In this case, a reference number will not be returned.

First, `AS_PlayID` attempts to load the specified `'snd '` resource. If successful, the handle state is saved for later restoration, and the handle is made unpurgeable. The function then gets a reference number and a pointer to the next free `ASStructure`. A sound channel is then allocated via a call to SndNewChannel and the associated `ASStructure` is initialised. `HLockHi` is then called to move the sound handle high in the heap and lock it. `SndPlay` is then called to start the sound playing, playing, the `channel.userInfo` field is set to indicate that the sound is playing, and a callback function is queued so that `AsynchSoundLib` will know when the sound has stopped playing. If all this is successful, `AS_PlayID` returns the reference number associated with the channel (if the caller wants it).

```
OSErr  AS_PlayHandle(sound,refNum);

Handle sound    A handle to the sound to be played.
SInt32 *refNum  A pointer to a reference number storage variable. Optional.

Returns:  0  If no errors.
          1  No channels available.
          Non-zero results of SndNewChannel call.
          Non-zero results of SndPlay call.
```

This function initiates asynchronous playback of the sound referred to by `sound`.

The `AS_PlayHandle` function is similar to `AS_PlayID`, except that it supports a special case: You can pass `AS_PlayHandle` a `NULL` handle. This causes `AS_PlayHandle` to open a sound channel but not call `SndPlay`. Normally, you do this when you want to get a sound channel and then send sound commands directly to that channel yourself. (See `AS_GetChannel`, below.)

If a handle is provided, its current state is saved for later restoration before it is made unpurgeable. `AS_PlayHandle` then gets a reference number and a pointer to a free `ASStructure`. A sound channel is then allocated via a call to `SndNewChannel` and the associated `ASStructure` is initialised. Then, if a handle was provided, `HLockHi` is called to move the sound handle high in the heap and lock it, following which `SndPlay` is called to start the sound playing, the `channel.userInfo` field is set to indicate that the sound is playing, and a callback function is queued so that `AsynchSoundLib` will know when the sound has stopped playing. Finally, the reference number associated with the channel is returned (if the caller wants it).

```
OSErr  AS_GetChannel(refNum,channel);

Sint32        refNum    Reference number.
SndChannelPtr *channel  A pointer to a SoundChannelPtr.

Returns:  0  No errors.
```

```
1  If refNum does not refer to any current ASStructure.
```

This function searches for the `ASStructure` associated with `refNum`. If one is found, a pointer to the associated sound channel is retuned in the `channel` parameter.

`AS_GetChannel` is provided so as to allow an application to gain access to the sound channel associated with a specified reference number and thus gain the potential for more control over the playback process. It allows an application to use `AsynchSoundLib` to handle sound channel management while at the same time retaining the ability to send sound commands to the channel. This is most commonly done to play looped continuous music, for which you will need to provide a sound resource with a loop and a sound command to install the music as a voice. First, you open a channel by calling `AS_PlayHandle`, specifying NULL in the first parameter. (This causes `AS_PlayHandle` to open a sound channel but not call `SndPlay`.) Armed with the returned reference number associated with that channel, you then call `AS_GetChannel` to get the `SndChannelPtr`, which you then pass as the first parameter in a call to `SndPlay`. Finally, you send a `freqCmd` command to the channel to start the music playing. The playback will keep looping until you send a `quietCmd` command to the channel.

```
void  AS_CloseChannel(void);
```

This function is called from the application's event loop if the application's "attention" flag is set. It clears the "attention" flag and then performs playback cleanup by iterating through the `ASStructures` looking for structures that are both in use (that is, the `inUse` field contains `true`) *and* complete (that is, the `channel.userInfo` field has been set by `AsyncSoundLib`'s callback function to indicate that the sound has stopped playing). It frees up such structures for later use and closes the associated sound channel.

```
void  AS_CloseDown(void);
```

`AS_CloseDown` checks that `AsynchSoundLib` was previously initialized, stops all current playback, calls `AS_CloseChannel` to close open sound channels, and disposes of the associated `ASStructures`.

Demonstration Program SoundAndSpeech Listing

```
//
//**************************************************************************************************
// SoundAndSpeech.c                                                          CARBON EVENT
MODEL
//
//**************************************************************************************************
//
// This program opens a modeless dialog containing five bevel button controls arranged in
// two groups, namely, a synchronous sound group and an asynchronous sound group. Clicking
on
// the bevel buttons causes sound to be played back or recorded as follows:
//
// •  Synchronous group:
//
//    •  Play sound resource.
//
//    •  Record sound resource (Mac OS 8/9 only).
//
//    •  Speak text string.
//
// •  Asynchronous group:
//
//    •  Play sound resource.
//
//    •  Speak text string.
//
// The asynchronous sound sections of the program utilise a special library called
// AsyncSoundLibPPC, which must be included in the CodeWarrior project.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  A 'DLOG' resource and associated 'DITL', 'dlgx', and 'dftb' resources (all purgeable).
//
// •  'CNTL' resources (purgeable) for the controls within the dialog.
//
// •  Two 'snd ' resources, one for synchronous playback (purgeable) and one for
asynchronous
//    playback (purgeable).
//
// •  Four 'cicn' resources (purgeable). Two are used to provide an animated display which
//    halts during synchronous playback and continues during asynchronous playback. The
//    remaining two are used by the bevel button controls.
//
// •  Two 'STR#' resources containing "speak text" strings and error message strings (all
//    purgeable).
//
// •  'hrct' and 'hwin' resources (purgeable) for balloon help.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// Each time it is invoked, the function doRecordResource creates a new 'snd' resource with
a
// unique ID in the resource fork of a file titled "SoundResources".
//
//
//**************************************************************************************************
```

```
//   ..............................................................................
includes

#include <Carbon.h>
#include <string.h>

// ..............................................................................
defines

#define rDialog                 128
#define  iDone                  1
#define  iPlayResourceSync      4
#define  iRecordResource        5
#define  iSpeakTextSync         6
#define  iPlayResourceASync     7
#define  iSpeakTextAsync        8
#define rPlaySoundResourceSync  8192
#define rPlaySoundResourceASync 8193
#define rSpeechStrings          128
#define rErrorStrings           129
#define  eOpenDialogFail        1
#define  eCannotInitialise      2
#define  eGetResource           3
#define  eMemory                4
#define  eMakeFSSpec            5
#define  eWriteResource         6
#define  eNoChannelsAvailable   7
#define  ePlaySound             8
#define  eSndPlay               9
#define  eSndRecord             10
#define  eSpeakString           11
#define rColorIcon1            128
#define rColorIcon2            129
#define kMaxChannels            8
#define kOutOfChannels          1

// ..................................................................... global
variables

Boolean     gRunningOnX = false;
DialogRef   gDialogRef;
CIconHandle gColorIconHdl1;
CIconHandle gColorIconHdl2;

// ..................................................... AsyncSoundLib attention
flag

Boolean  gCallAS_CloseChannel = false;
// ..................................................................function
prototypes

void      main                (void);
void      doPreliminaries     (void);
OSStatus  windowEventHandler  (EventHandlerCallRef,EventRef,void *);
void      doIdle              (void);
void      doInitialiseSoundLib (void);
void      doDialogHit         (SInt16);
void      doPlayResourceSync  (void);
void      doRecordResource    (void);
void      doSpeakStringSync   (void);
void      doPlayResourceASync (void);
void      doSpeakStringAsync  (void);
void      doSetUpDialog       (void);
void      doErrorAlert        (SInt16);
void      helpTags            (DialogRef);
```

```
// ..........................................................AsyncSoundLib function
prototypes

OSErr  AS_Initialise   (Boolean *,SInt16);
OSErr  AS_GetChannel   (SInt32,SndChannelPtr *);
OSErr  AS_PlayID       (SInt16, SInt32 *);
OSErr  AS_PlayHandle   (Handle,SInt32 *);
void   AS_CloseChannel (void);
void   AS_CloseDown    (void);

// ****************************************************************************
main

void main(void)
{
  SInt32         response;
  EventTypeSpec  windowEvents[] = { { kEventClassWindow, kEventWindowClose } ,
                                    { kEventClassMouse,  kEventMouseDown   } };

  // ..................................................................do
preliminaries

  doPreliminaries();

  // ..................................... disable Quit item in Mac OS X Application
menu

  DisableMenuCommand(NULL,'quit');

  // ..................................................................install a
timer

  InstallEventLoopTimer(GetCurrentEventLoop(),0,TicksToEventTime(10),
                        NewEventLoopTimerUPP((EventLoopTimerProcPtr) doIdle),NULL,
                        NULL);

  // ..........................................................open and set up
dialog

  if(!(gDialogRef = GetNewDialog(rDialog,NULL,(WindowRef) -1)))
  {
    doErrorAlert(eOpenDialogFail);
    ExitToShell();
  }
  SetPortDialogPort(gDialogRef);
  SetDialogDefaultItem(gDialogRef,kStdOkItemIndex);

  ChangeWindowAttributes(GetDialogWindow(gDialogRef),kWindowStandardHandlerAttribute |
                                                     kWindowCloseBoxAttribute,
                                                     kWindowCollapseBoxAttribute);

  InstallWindowEventHandler(GetDialogWindow(gDialogRef),
                            NewEventHandlerUPP((EventHandlerProcPtr) windowEventHandler),

                            GetEventTypeCount(windowEvents),windowEvents,0,NULL);

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    helpTags(gDialogRef);
    gRunningOnX = true;
  }

  doSetUpDialog();

  // ..........................................................initialise
```

```
AsyncSoundLib

  doInitialiseSoundLib();

  // ..................................................................get color icons

  gColorIconHdl1 = GetCIcon(rColorIcon1);
  gColorIconHdl2 = GetCIcon(rColorIcon2);

  // ..........................................................run application event loop

  RunApplicationEventLoop();
}

// ************************************************************************
doPreliminaries

void  doPreliminaries(void)
{
  MoreMasterPointers(64);
  InitCursor();
  FlushEvents(everyEvent,0);
}

// ************************************************************************
windowEventHandler

OSStatus  windowEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                             void* userData)
{
  OSStatus    result = eventNotHandledErr;
  UInt32      eventClass;
  UInt32      eventKind;
  EventRecord eventRecord;
  SInt16      itemHit;
  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);

  switch(eventClass)
  {
    case kEventClassWindow:                                          // event class
window
      switch(eventKind)
      {
        case kEventWindowClose:
          AS_CloseDown();
          QuitApplicationEventLoop();
          result = noErr;
          break;
      }

    case kEventClassMouse:                                           // event class
mouse
      ConvertEventRefToEventRecord(eventRef,&eventRecord);
      switch(eventKind)
      {
        case kEventMouseDown:
          if(IsDialogEvent(&eventRecord))
          {
            if(DialogSelect(&eventRecord,&gDialogRef,&itemHit))
              doDialogHit(itemHit);
            result = noErr;
          }
          break;
      }
      break;
```

```
  }

  return result;
}

// ****************************************************************************
doIdle

void  doIdle(void)
{
  Rect           theRect, eraseRect;
  UInt32         finalTicks;
  SInt16         fontNum;
  static Boolean flip;

  SetRect(&theRect,262,169,294,201);
  SetRect(&eraseRect,310,170,481,200);

  if(gCallAS_CloseChannel)
  {
    AS_CloseChannel();

    GetFNum("\pGeneva",&fontNum);
    TextFont(fontNum);
    TextSize(10);
    MoveTo(341,189);
    DrawString("\pAS_CloseChannel called");
    QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
    Delay(45,&finalTicks);
  }

  if(flip)
    PlotCIcon(&theRect,gColorIconHdl1);
  else
    PlotCIcon(&theRect,gColorIconHdl2);

  flip = !flip;

  EraseRect(&eraseRect);
}

// ****************************************************************
doInitialiseSoundLib

void  doInitialiseSoundLib(void)
{
  if(AS_Initialise(&gCallAS_CloseChannel,kMaxChannels) != noErr)
  {
    doErrorAlert(eCannotInitialise);
    ExitToShell();
  }
}

// ****************************************************************************
doDialogHit

void  doDialogHit(SInt16 item)
{
  switch(item)
  {
    case iDone:
      AS_CloseDown();
      QuitApplicationEventLoop();
      break;

    case iPlayResourceSync:
```

```
      doPlayResourceSync();
      break;

    case iRecordResource:
      doRecordResource();
      break;

    case iSpeakTextSync:
      doSpeakStringSync();
      break;

    case iPlayResourceASync:
      doPlayResourceASync();
      break;
    case iSpeakTextAsync:
      doSpeakStringAsync();
      break;
  }
}

// ************************************************************************
doPlayResourceSync

void  doPlayResourceSync(void)
{
  SndListHandle sndListHdl;
  SInt16        resErr;
  OSErr         osErr;
  ControlRef    controlRef;

  sndListHdl = (SndListHandle) GetResource('snd ',rPlaySoundResourceSync);
  resErr = ResError();
  if(resErr != noErr)
    doErrorAlert(eGetResource);

  if(sndListHdl != NULL)
  {
    HLock((Handle) sndListHdl);
    osErr = SndPlay(NULL,sndListHdl,false);
    if(osErr != noErr)
      doErrorAlert(eSndPlay);
    HUnlock((Handle) sndListHdl);
    ReleaseResource((Handle) sndListHdl);

    GetDialogItemAsControl(gDialogRef,iPlayResourceSync,&controlRef);
    SetControlValue(controlRef,0);
  }
}

// ************************************************************************
doRecordResource

void  doRecordResource(void)
{
  SInt16     oldResFileRefNum, theResourceID, resErr, tempResFileRefNum;
  BitMap     screenBits;
  Point      topLeft;
  OSErr      memErr, osErr;
  Handle     soundHdl;
  FSSpec     fileSpecTemp;
  ControlRef controlRef;

  oldResFileRefNum = CurResFile();

  GetQDGlobalsScreenBits(&screenBits);
  topLeft.h = (screenBits.bounds.right / 2) - 156;
```

```
  topLeft.v = 150;

  soundHdl = NewHandle(25000);
  memErr = MemError();
  if(memErr != noErr)
  {
    doErrorAlert(eMemory);
    return;
  }

  osErr = FSMakeFSSpec(0,0,"\pSoundResources",&fileSpecTemp);
  if(osErr == noErr)
  {
    tempResFileRefNum = FSpOpenResFile(&fileSpecTemp,fsWrPerm);
    UseResFile(tempResFileRefNum);
  }
  else
    doErrorAlert(eMakeFSSpec);

  if(osErr == noErr)
  {
    osErr = SndRecord(NULL,topLeft,siBetterQuality,&(SndListHandle) soundHdl);
    if(osErr != noErr && osErr != userCanceledErr)
      doErrorAlert(eSndRecord);
    else if(osErr != userCanceledErr)
    {
      do
      {
        theResourceID = UniqueID('snd ');
      }  while(theResourceID <= 8191 && theResourceID >= 0);
      AddResource(soundHdl,'snd ',theResourceID,"\pTest");
      resErr = ResError();
      if(resErr == noErr)
        UpdateResFile(tempResFileRefNum);
      resErr = ResError();
      if(resErr != noErr)
        doErrorAlert(eWriteResource);
    }

    CloseResFile(tempResFileRefNum);
  }

  DisposeHandle(soundHdl);
  UseResFile(oldResFileRefNum);

  GetDialogItemAsControl(gDialogRef,iRecordResource,&controlRef);
  SetControlValue(controlRef,0);
}

// ************************************************************************
doSpeakStringSync

void  doSpeakStringSync(void)
{
  SInt16     activeChannels;
  Str255     theString;
  OSErr      resErr, osErr;
  ControlRef controlRef;
  activeChannels = SpeechBusy();

  GetIndString(theString,rSpeechStrings,1);
  resErr = ResError();
  if(resErr != noErr)
  {
    doErrorAlert(eGetResource);
    return;
```

```
  }

  osErr = SpeakString(theString);
  if(osErr != noErr)
    doErrorAlert(eSpeakString);

  while(SpeechBusy() != activeChannels)
    ;

  GetDialogItemAsControl(gDialogRef,iSpeakTextSync,&controlRef);
  SetControlValue(controlRef,0);
}

// ************************************************************************
doPlayResourceASync

void  doPlayResourceASync(void)
{
  SInt16 error;

  error = AS_PlayID(rPlaySoundResourceASync,NULL);
  if(error == kOutOfChannels)
    doErrorAlert(eNoChannelsAvailable);
  else
    if(error != noErr)
      doErrorAlert(ePlaySound);
}

// ************************************************************************
doSpeakStringAsync

void  doSpeakStringAsync(void)
{
  Str255 theString;
  OSErr  resErr, osErr;

  GetIndString(theString,rSpeechStrings,2);
  resErr = ResError();
  if(resErr != noErr)
  {
    doErrorAlert(eGetResource);
    return;
  }

  osErr = SpeakString(theString);
  if(osErr != noErr)
    doErrorAlert(eSpeakString);
}
// ************************************************************************
doSetUpDialog

void  doSetUpDialog(void)
{
  SInt16                      a;
  Point                       offset;
  ControlRef                  controlRef;
  ControlButtonGraphicAlignment alignConstant = kControlBevelButtonAlignLeft;
  ControlButtonTextPlacement    placeConstant = kControlBevelButtonPlaceToRightOfGraphic;

  offset.v = 1;
  offset.h = 5;

  for(a=iPlayResourceSync;a<iSpeakTextAsync+1;a++)
  {
    GetDialogItemAsControl(gDialogRef,a,&controlRef);
    SetControlData(controlRef,kControlEntireControl,kControlBevelButtonGraphicAlignTag,
```

```
                  sizeof(alignConstant),&alignConstant);
    SetControlData(controlRef,kControlEntireControl,kControlBevelButtonGraphicOffsetTag,
                  sizeof(offset),&offset);
    SetControlData(controlRef,kControlEntireControl,kControlBevelButtonTextPlaceTag,
                 sizeof(placeConstant),&placeConstant);
  }

  if(gRunningOnX)
  {
    GetDialogItemAsControl(gDialogRef,iRecordResource,&controlRef);
    DeactivateControl(controlRef);
  }
}

// ************************************************************************
doErrorAlert

void  doErrorAlert(SInt16 errorStringIndex)
{
  Str255 errorString;
  SInt16 itemHit;

  GetIndString(errorString,rErrorStrings,errorStringIndex);
  StandardAlert(kAlertCautionAlert,errorString,NULL,NULL,&itemHit);
}

// ************************************************************************************
helpTags

void  helpTags(DialogRef dialogRef)
{
  HMHelpContentRec helpContent;
  SInt16           a;
  ControlRef       controlRef;

  memset(&helpContent,0,sizeof(helpContent));
  HMSetTagDelay(500);
  HMSetHelpTagsDisplayed(true);
  helpContent.version = kMacHelpVersion;
  helpContent.tagSide = kHMOutsideTopCenterAligned;
  helpContent.content[kHMMinimumContentIndex].contentType = kHMStringResContent;
  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmResID = 130;

  for(a = 1;a <= 5; a++)
  {
    if(a == 2)
      continue;
    helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = a;
    GetDialogItemAsControl(dialogRef,a + 3,&controlRef);
    HMSetControlHelpContent(controlRef,&helpContent);
  }
}

//
**********************************************************************************************
```

Demonstration Program SoundAndSpeech Comments

When this program is run, the user should click on the various buttons in the dialog to play back and record (Mac OS 8/9 only) sound resources and to play back the provided "speak text" strings. The user should observe the effects of asynchronous and synchronous playback on the "working man" icon in the image well in the dialog. The user should also observe that the text "`AS_CloseChannel called`" appears briefly in the secondary group box to the right of the "working man" icon when `AsynchSoundLib` sets the application's "attention" flag to true, thus causing the application to call the `AsynchSoundLib` function `AS_CloseChannel`.

Note that the `doRecordResource` function saves recorded sounds as `'snd '` resources with unique IDs in the resource fork of the file titled "SoundResources."

On Mac OS 9, ensure that the Speech Manager extension is activated before running this program.

Defines

`kMaxChannels` will be used to specify the maximum number of sound channels that `AsynchSoundLib` is to open. `kOutOfChannels` will be used to determine whether the `AsynchSoundLib` function `AS_PlayID` returns a "no channels available" error.

Main

A timer is installed and set to fire repeatedly every 10 ticks. When the timer fires, the function `doIdle` is called.

`doInitialiseSoundLib` is called to initialise the `AsynchSoundLib` library.

doIdle

`doIdle` is called every time the timer fires.

The "attention" flag (`gAS_CloseChannel`) required by `AsynchSoundLib` is checked. If `AsynchSoundLib` has set it to true, the `AsynchSoundLib` function `AS_CloseChannel` is called to free up the relevant `ASStructure`, close the relevant sound channel, and clear the "attention" flag. In addition, some text is drawn in the group box to the right of the "working man" icon to indicate to the user that `AS_CloseChannel` has just been called.

The next block draws one or other of the two "working man" icons, following which the interior of the group box is erased.

doInitialiseSoundLib

`doInitialiseSoundLib` initializes the `AsynchSoundLib` library. More specifically, it calls the `AsynchSoundLib` function `AS_Initialise` and passes to `AsynchSoundLib` the address of the application's "attention" flag (`gAS_CloseChannel`), together with the requested number of channels.

If `AS_Initialise` returns a non-zero value, an error alert is displayed and the program terminates.

doPlayResourceSync

`doPlayResourceSync` is the first of the synchronous playback functions. It uses `SndPlay` to play a specified `'snd '` resource.

`GetResource` attempts to load the resource. If the subsequent call to ResError indicates an error, an error alert is presented.

If the load was successful, the sound handle is locked prior to a call to `SndPlay`. Since `NULL` is passed in the first parameter of the `SndPlay` call, `SndPlay` automatically allocates a sound channel to play the sound and deallocates the channel when the playback is complete. `false` passed in the third parameter specifies that the playback is to be synchronous.

> **Note**
>
> The 39940-byte `'snd '` resource being used contains one command only (`bufferCmd`). The compressed sound header indicates MACE 3:1 compression. The sound length is 119568 frames. The 8-bit mono sound was sampled at 22kHz.

`SndPlay` causes all commands and data contained in the sound handle to be sent to the channel. Since there is a `bufferCmd` command in the `'snd '` resource, the sound is played.

If `SndPlay` returns an error, an error alert is presented.

When `SndPlay` returns, `HUnlock` unlocks the sound handle and `ReleaseResource`

releases the resource.

doRecordResource

On Mac OS 8/9 only, `doRecordResource` uses `SndRecord` to record a sound synchronously and then saves the sound in a `'snd '` resource. The `'snd '` resource will be saved to the resource fork of the file "SoundResources."

The first line saves the file reference number of the current resource file. The next three lines establish the location for the top left corner of the sound recording dialog.

`NewHandle` creates a relocatable block. The address of the handle will be passed as the fourth parameter of the `SndRecord` call. The size of this block determines the recording time available. (If `NULL` is passed as the fourth parameter of a `SndRecord` call, the Sound Manager allocates the largest block possible in the application's heap.) If `NewHandle` cannot allocate the block, an error alert is presented and the function returns.

The next block opens the resource fork of the file "SoundResources" and makes it the current resource file.

`SndRecord` opens the sound recording dialog and handles all user interaction until the user clicks the Cancel or Save button. Note that the second parameter of the `SndRecord` call establishes the location for the top left corner of the sound recording dialog and that the third parameter specifies 22kHz, mono, 3:1 compression.

When the user clicks the Save button, the handle is resized automatically. If the user clicks the Cancel button, `SndRecord` returns `userCanceledErr`. If `SndRecord` returns an error other than `userCanceledErr`, an error alert is presented and the function returns after closing the resource fork of the file, disposing of the relocatable block, and restoring the saved resource file reference number.

The relocatable block allocated by `NewHandle`, and resized as appropriate by `SndPlay`, has the structure of a `'snd '` resource, but its handle is not a handle to an existing resource. To save the recorded sound as a `'snd '` resource in the resource fork of the current resource file, the do/while loop first finds an acceptable unique resource ID for the resource. (For the System file, resource IDs for `'snd '` resources in the range 0 to 8191 are reserved for use by Apple Computer, Inc. Avoiding those IDs in this demonstration is not strictly necessary, since there is no intention to move those resources to the System file.)

The call to `AddResource` causes the Resource Manager to regard the relocatable block containing the sound as a `'snd '` resource. If the call is successful, `UpdateResFile` writes the changed resource map and the `'snd '` resource to disk. If an error occurs, an error alert is presented.

The relocatable block is then disposed of, the resource fork of the file "SoundResources" is closed, and the saved resource file reference number is restored.

doSpeakStringSync

`doSpeakStringSync` uses `SpeakString` to speak a specified string resource and takes measures to cause the speech to be generated in a psuedo-synchronous manner.

The speech that `SpeakString` generates is asynchronous, that is, control returns to the application before `SpeakString` finishes speaking the string. In this function, `SpeechBusy` is used to cause the speech activity to be synchronous so far as the function as a whole is concerned. That is, `doSpeakStringSync` will not return until the speech activity is complete.

As a first step, the first line saves the number of speech channels that are active immediately before the call to `SpeakString`.

`GetIndString` loads the first string from the specified `'STR#'` resource. If an error occurs, an error alert is presented and the function returns.

`SpeakString`, which automatically allocates a speech channel, is called to speak the string. If `SpeakString` returns an error, an error alert is presented.

Although `SpeakString` returns control to the application immediately it starts generating the speech, the speech channel it opens remains open until the speech concludes. While the speech continues, the number of speech channels open will be one more that the number saved at the first line. Accordingly, the while loop continues until the number of open speech channels is equal to the number saved at the first line. Then, and only then, does `doSpeakStringSync` exit.

doPlayResourceASync

`doPlayResourceASync` uses the AsynchSoundLib function `AS_PlayID` to play a `'snd '` resource asynchronously.

> **Note**
>
> The 24194-byte `'snd '` resource being used contains one command only (`bufferCmd`). The compressed sound header indicates no compression. The sound length is 24195 frames. The 8-bit mono sound was sampled at 5kHz.

AS_PlayID is called to play the 'snd ' resource specified in the first parameter. Since no further control over the playback is required, NULL is passed in the second parameter. (Recall that, if you pass a pointer to a variable in the second parameter, AS_PlayID returns a reference number in that parameter. That reference number may be used to gain more control over the playback process. If you simply want to trigger a sound and let it to run to completion, you pass NULL in the second parameter, in which case a reference number is not returned by AS_PlayID.)

If AS_PlayID returns the "no channels currently available" error, an error alert is presented advising of that specific condition. If any other error is returned, a more generalized error message is presented.

When the sound has finished playing, ASynchSoundLib advises the application by setting the application's "attention" flag to true. Recall that this will cause the AsynchSoundLib function AS_CloseChannel to be called to free up the relevant ASStructure, close the relevant sound channel, clear the "attention" flag, and draw some text in the group box to the right of the image well to indicate to the user that AS_CloseChannel has just been called.

doSpeakStringAsync

doSpeakStringAsync is identical to the function doSpeakStringSync except that, in this function, SpeechBusy is not used to delay the function returning until the speech activity spawned by SpeakString has run its course.

doSetUpDialog

Within doSetUpDialog, the Record Sound Resource bevel button is disabled if the program is running on OS X.

Chapter 25. Miscellany

IN THIS CHAPTER

- Notification from Applications in the Background
- **Progress Bars Command-Period Key-Down Events, and Mouse-Down Events**
- The Color Picker
- Coping with Multiple Monitors
- Help Tags
- Ensuring Compatibility with the Operating Environment
- Carbon and Available APIs
- Main Notification Manager Data Types and Functions
- Relevant Process Manager Data Types and Functions
- Relevant Event Manager Function
- Relevant Color Picker Utilities Function
- Relevant QuickDraw Function
- Relevant Window Manager Function
- Main Help Package Constants, Data Types, and Functions
- Relevant Gestalt Manager Function
- Demonstration Program Miscellany Listing
- Demonstration Program Miscellany Comments

Notification from Applications in the Background

The Need for the Notification Manager

Applications running in the background sometimes need to communicate information to the user and need to be certain that the user has actually received that information. The Notification Manager provides for this requirement.

Elements of a Notification

The Notification Manager creates **notifications.** A notification comprises one or more of five possible elements, which occur in the following sequence:

- A mark appears against the name of the target application in the Mac OS 8/9 Application menu.

  This mark is intended to prompt the user to switch the marked application to the foreground. The _ mark only appears while the application posting the notification remains in the background. It is replaced by the familiar mark when that application is brought to the foreground.

- The Mac OS 8/9 Application menu begins alternating between the target application's icon and the foreground application's icon, or the Apple menu title begins alternating between the target application's icon and the Apple icon.

  The location of the icon alternation in the Mac OS 8/9 menu bar is determined by the posting application's mark (if any). If the application posting the notification is marked by either a _ mark or a _ mark in the Mac OS 8/9 Application menu, the Application menu title alternates; otherwise the Apple menu title alternates.

  Note that several applications might post notifications, so there might be a series of alternating icons.

- On Mac OS 8/9, the Sound Manager plays a sound.

  The application posting the notification can request that the system alert sound be used or it can specify its own sound by passing the Notification Manager a handle to a `'snd '` resource.

- On Mac OS 8.6, a modal alert appears, and the user dismisses it (by clicking on the Cancel button). On Mac OS 9.x a floating window appears, allowing the user to continue working in any running application without first dismissing the notification window. On Mac OS X, a system movable modal alert, which remains in front of all other windows, appears.

  The application posting the notification specifies the text for the modal alert/floating window/ system movable modal alert.

- A response function, if specified, executes.

  A response function can be used to remove the notification request from the notification queue (see the following section) or to perform other processing. For example, it can be used to set a global variable to record that the notification was received.

Notifications in Action

Overview

The Notification Manager is automatically initialized at system startup.

To present the user with a notification, you create a notification request and install it into the notification queue, which is a standard Macintosh queue. The Notification Manager presents the notification to the user at the earliest possible time.

When appropriate (that is, in the response function or when your application returns to the foreground), you can remove the notification request from the notification queue.

Creating a Notification Request

The Notification Structure

When installing a request into the notification queue, your application must supply a pointer to a notification structure, a static and nonrelocatable structure of type `NMRec`, which indicates the type of notification required. Each entry in the notification queue is a notification structure. The notification structure is as follows:

```
struct NMRec
{
  QElemPtr  qLink;       // Address of next element in queue. (Used internally.)
  short     qType;       // Type of data. (8 = nmType).
  short     nmFlags;     // (Reserved.)
  long      nmPrivate;   // (Reserved.)
  short     nmReserved;  // (Reserved.)
  short     nmMark;      // Application to identify with diamond mark.
  Handle    nmIcon;      // Handle to small icon.
  Handle    nmSound;     // Handle to sound structure.
  StringPtr nmStr;       // Pointer to string to appear in the notification.
  NMUPP     nmResp;      // Pointer to response function.
  long      nmRefCon;    // Available for application use.
} ;
typedef struct NMRec NMRec;
typedef NMRec *NMRecPtr;
```

Field Descriptions

To set up a notification request, you need to fill in at least the first six of the following fields:

- `qType` The type of operating system queue. Set to `nmType` (8).
- `nmMark` Specifies whether to place a • mark next to the name of the application in the Mac OS 8/9 Application menu. `0` means no mark appears. `1` means the mark appears. Applications should ordinarily set this field to `1`.
- `nmIcon` A handle to an icon family containing a small color icon that is to alternate periodically in the menu bar. If this field is set to `NULL`, no icon appears. The handle must be non-purgeable.
- `nmSound` A handle to a sound resource. If this field is set to `NULL`, no sound is played. If this field is set to `-1`, the system alert sound is played. The handle must be non-purgeable.

- `nmStr` Pointer to a string that appears in the alert/floating window/system movable modal alert. If this field is set to `NULL`, no alert/ floating window/system movable modal alert appears. Your application should not dispose of this storage until it removes the notification request.

- `nmResp` Universal procedure pointer to a response function. If this field is set to `NULL`, no response function executes when the notification is posted. If this field is set to `-1`, a pre-defined function removes the notification request when it has completed. However, on Mac OS 8/9, if either `nmMark` or `nmIcon` is non-zero, do not set `nmResp` to `-1`, because the Notification Manager will remove the _ mark or the icon before the user sees it.

- If you do not need to do any processing in response to the notification, you should set this field to `NULL`. If you supply a universal procedure pointer to your own response function, the Notification Manager passes your response function one parameter, namely, a universal procedure pointer to your notification structure. Accordingly, this is how you would declare a response function having the name `theResponse`:

  ```
  void  theResponse(NMUPP nmStructurePtr);
  ```

- You can use response functions to remove notification requests from the notification queue, free any memory, or set a global variable in your application to record that the notification was posted.
- An *`nmResp`* value of *`-1`* does not free the memory block containing the queue element; it merely removes that element from the notification queue.
- `nmRefCon` For your application's own use.

Installing a Notification Request

`NMInstall` is used to add a notification request to the notification queue. The following is an example call:

```
osError = NMInstall(&notificationStructure);
```

Before calling `NMInstall`, you should make sure that your application is running in the background. If your application is in the foreground, you simply use standard alert methods, rather than the Notification Manager, to gain the user's attention.

Removing a Notification Request

`NMRemove` is used to remove a notification request from the notification queue. The following is an example call:

```
osError = NMRemove(&notificationStructure);
```

You can remove `requests` at any time, either before or after the notification actually occurs.

On Mac OS 9.x and Mac OS X the user does not have to dismiss the notification before being able to activate the application. For this reason, when your application is running on Mac OS 9.x or Mac OS X, may wish to have it explicitly cancel the notification using `NMRemove` when the application becomes active.

Progress Bars, Command-Period Key-Down Events, and Mouse-Down Events

Progress Bars

Operations within an application that tie up the machine for relatively brief periods of time should be accompanied by a cursor shape change to the watch cursor or perhaps to an animated cursor (Mac OS 8/9) or by an invocation of the wait cursor (Mac OS X). On the other hand, lengthy operations should be accompanied by the display of a progress indicator.

The progress indicator control was described in Chapter 14. A progress indicator created using this control may be determinate or indeterminate. Determinate progress indicators show how much of the operation has been completed. Indeterminate progress indicators show that an operation is occurring but does not indicate its duration. Ordinarily, progress indicators should be displayed within a dialog.

As stated in Chapter 2, your application should allow the user to cancel a lengthy operation using the Command-period key combination.

Scanning for Command-Period Key-Down Events

The function `CheckEventQueueForUserCancel` may be used to scan the event queue for Command-period key-down events, and will return `true` if a Command-period event is found. If `true` is returned, the lengthy operation should be terminated and the dialog displaying the progress indicator should be closed.

The Color Picker

The Color Picker Utilities provide your application with:

- A standard dialog, called the Color Picker, for soliciting a color choice from the user.
- Functions for converting color specifications from one color model to another.

Color Models

In the world of color, three main color models are used to specify a particular color. These are the RGB (red, green, blue) model, the CYMK (cyan, magenta, yellow, black) model, and the HLS or HSV (hue, lightness, saturation, or hue, saturation, value) models.

RGB Model

The RGB model is used where light-produced colors are involved, as in the case of a television set, computer monitor, or stage lighting. In this model, the three primary colors involved (red, green, and blue) are said to be *additive* because, the more of each color you add, the closer the resulting color is to white.

CYMK Model

The CYMK model is closely associated with printing, that is, putting color on a white page. In this model, the three primary colors (cyan, yellow, and magenta) are said to be *subtractive* because, the more of each color you add, the closer the resulting color is to black. (The inclusion of black in the model accounts for the fact that the colors of printers' inks may vary slightly from true cyan, yellow, and magenta, meaning that a true black may not be achievable with just a CYM model.)

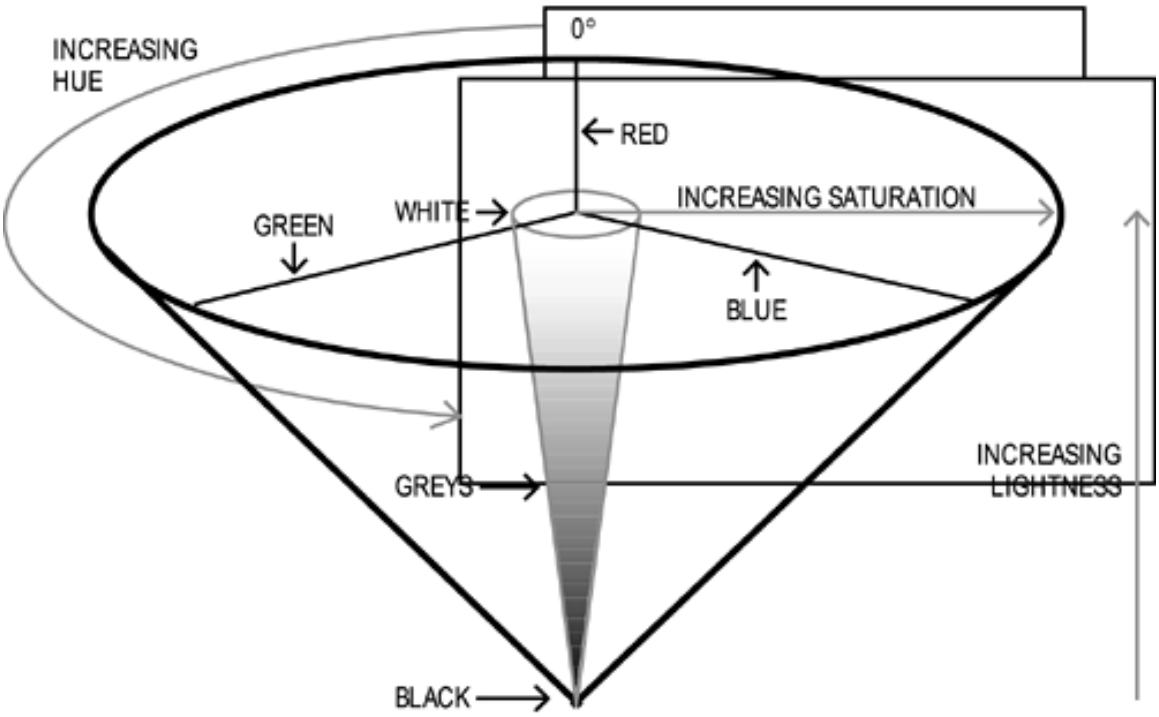
> **Note**
>
> Cyan, magenta, and yellow are the complements of red, green, and blue.

HLS and HSV Models

The HLS and HSV models separate color (that is, hue) from saturation and brightness. Saturation is a measure of the amount of white in a color (the less white, the more saturated the color). Lightness is the measure of the amount of black in a color. (The less black, the lighter the color). The amount of black is specified by the lightness (L) value in the HLS model and by the value (V) value in the HSV model.

The HSL/HLV model may be represented diagrammatically by the HSL/HLV color cone shown in Fig 1. In this color cone, hue is represented by an angle between 0 Þ and 360 Þ.

Figure 1. HSL/HSV COLOUR CONE



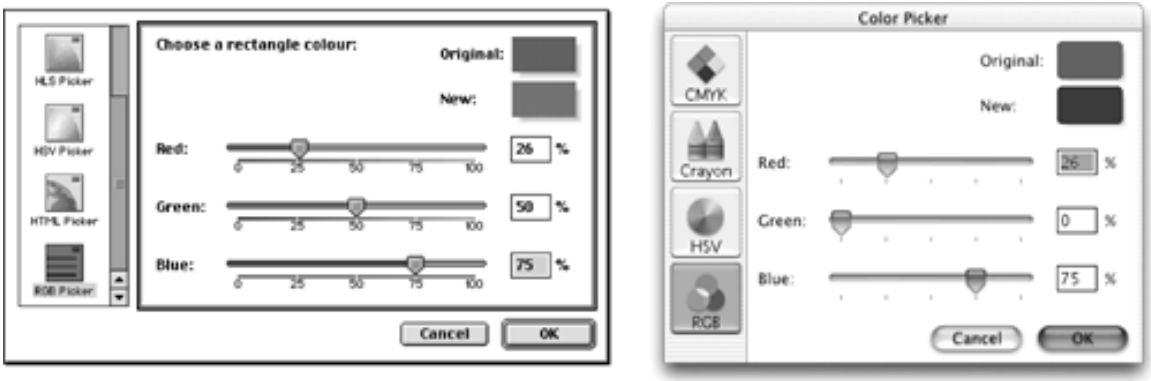
The Color Picker

The Color Picker allows the user to specify a color using either the RGB, CMYK, HLS, or HSV, models.

Using the Color Picker RGB Mode

Fig 2 shows the Color Picker in RGB mode. The desired red, green, and blue values may be set using the three slider controls or may be entered directly into the edit text fields on the right of the sliders.

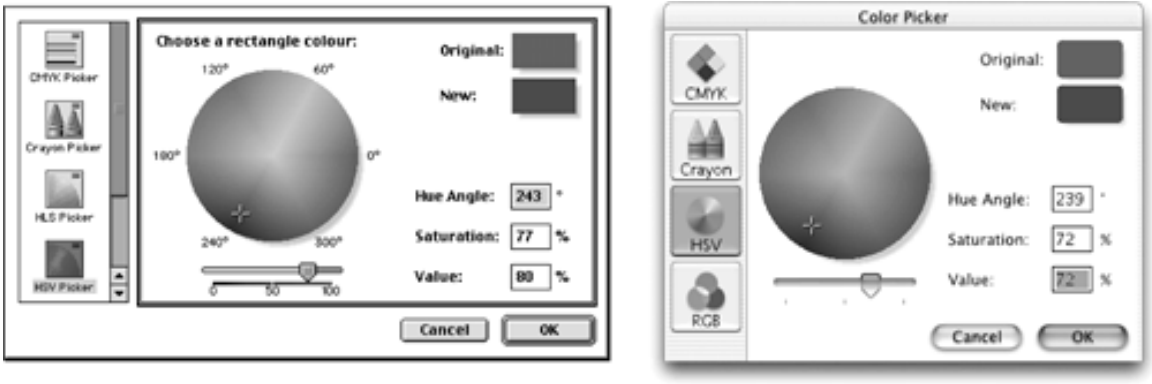
Figure 2. COLOR PICKER DIALOG IN RGB MODE



Using the Color Picker in HSV Mode

Fig 3 shows the Color Picker in HSV mode. Hue is specified by an angle, which may be entered at Hue Angle:. Saturation is specified by percentage, which may be entered at Saturation:. Value is also specified by a percentage, which may be entered at Value:. Alternatively, hue and saturation may be selected simultaneously by clicking at the desired point within the colored disc, and value may be set with the slider control.

Figure 3. COLOR PICKER DIALOG IN HLS MODE



To relate Fig 3 to Fig 1, the colored disc in Fig 3 may be considered as the HSL/HSV cone as viewed from above. The value slider control can then be conceived of as moving the disc up or down the axis of the cone from the apex (black) to the base (white).

Invoking the Color Picker

The Color Picker is invoked using the `GetColor` function:

```
Boolean GetColor(Point where,ConstStr255Param prompt,const RGBColor *inColor,
                 RGBColor *outColor);
```

Returns: A `Boolean` value indicating whether the user clicked on the OK button or Cancel button.

`Where` Dialog's upper-left corner. `(0,0)` causes the dialog to positioned centrally on the main screen.

`prompt` A prompt string, which is displayed in the upper left corner of the main pane in the dialog.

`inColor` The starting color, which the user may want for comparison, and which is displayed against Original: in the top-right corner of the dialog.

`outColor` Initially set to equal `inColor`. Assigned a new value when the user picks a color. The color stored in this parameter is displayed at the top right of the dialog against New:.)

If the user clicks the OK button in the Color Picker dialog, your application should adopt the `outColor` value as the color chosen by the user. If the user clicks the Cancel button, your application should assume that the user has decided to make no color change, that is, the color should remain as that represented by the `inColor` parameter.

Coping with Multiple Monitors

Overview

In a multi-monitor system, the use may specify which of the attached monitors is to be the main screen (that is, the screen containing the menu bar) and to set the position of the other screen, or screens, relative to the main screen.

The maximum number of colors capable of being displayed by a given Macintosh at the one time is determined by the video capability of that particular Macintosh. The maximum number of colors capable of being displayed on a given screen at the one time depends on settings made by the user. In a multi-monitor environment, therefore, it is possible for each screen to be set to a different pixel depth.

In more technical terms, the user's settings set the pixel depth of a particular video device. A brief review of the subject of video devices is therefore appropriate at this point.

Video Devices Revisited

As stated in Chapter 11

- A **graphics device** is anything into which QuickDraw can draw, a **video device** (such as a plug-in video card or a built-in video interface) is a graphics device that controls screens, QuickDraw stores information about video devices in `GDevice` structures, the system creates and initializes a `GDevice` structure for each video device found during start-up all structures are linked together in a list called the **device list,** and the global variable `DeviceList` holds a handle to the first structure in the list.
- At any given time only one graphics device is the **current device,** that is, the one in which the drawing is taking place. A handle to the current device's `GDevice` structure is placed in the global variable `TheGDevice`.

  Note

  The current device is sometimes referred to as the **active device.**

By default, the `GDevice` structure corresponding to the first video device found at start up is marked as the (initial) current device, and all other graphics devices in the list are initially marked as inactive. When the user moves a window to, or creates a window on, another screen, and your application draws into that window, QuickDraw automatically makes the video device for that screen the current device and stores that information in `TheGDevice`. As QuickDraw draws across a user's video devices, it keeps switching to the `GDevice` structure for the video device on which it is actively drawing.

Requirements of the Application

Image Optimization

To draw a particular graphic, your application may have to call different drawing functions for that graphic depending on the characteristics of the video device intersecting your window's drawing region, the aim being to

optimize the appearance of the image regardless of whether it is being displayed on a grayscale device or a color device. Recall from Chapter 11 that when QuickDraw displays a color on a grayscale screen, it computes the luminance, or intensity of light, of the desired color and uses that value to determine the appropriate gray value to draw. It is thus possible that, for example, two overlapping objects drawn in two quite different colors on a color screen may appear in the same shade of gray on a grayscale screen. In order for the user to differentiate between these two objects on a grayscale screen, you would need to provide an alternative drawing function which draws the two objects in different shades of gray on grayscale screens.

The QuickDraw function `DeviceLoop` is central to the matter of optimizing the appearance of your images. `DeviceLoop` searches for graphics devices that intersect your window's drawing region, informing your application of each graphics device it finds, and providing your application with information about the current device's attributes. Armed with this information, your application can then invoke whichever of its drawing functions is optimized for those particular attributes.

`DeviceLoop`'s second parameter is a pointer to an application-defined (callback) function. That function must be defined like this:

```
void  myDrawingFunction(short depth,short deviceFlags,GDHandle targetDevice,
                        long userData)
```

`DeviceLoop` calls this function for each dissimilar video device it finds. If it encounters similar devices (that is, devices having the same pixel depth, color table seeds, etc.) it will make only one call to `myDrawingFunction`, pointing to the first such device encountered.

Other Requirements-Classic Event Model Applications

Other requirements, for Classic event model applications only, are as follows:

- **Window Zooming .** If the user drags a window currently zoomed to the user state so that it spans two screens, and then clicks the zoom box to zoom the window to the standard state, the window should be zoomed to the standard state on the screen that contained the largest area of the window before the zoom box was clicked. The function `ZoomWindowIdeal`, which was introduced with Mac OS 8.5, zooms windows in accordance with this requirement.

- **Window Dragging.** In Carbon, if `NULL` is passed in `DragWindow`'s `boundsRect` parameter, the bounding rectangle limiting the area in which the window can be dragged is set to the desktop region, which, in a multi-monitors environment, includes all screen real estate less the menu bar.

- **Window Sizing.** In a multi-monitor environment, if you pass a constraining rectangle in `ResizeWindow`'s `sizeContraints` parameter, that rectangle should be based on the bounding rectangle of the desktop region. You should call `GetGrayRgn` to get a handle to the desktop region and then call `GetRegionBounds` to get that region's bounding rectangle.

Constraining a Window to One Screen

In a multimonitors environment, a call to the function `ConstrainWindowToScreen` enables you to constrain a window's movement and resizing so that it is contained entirely on a single screen.

Help Tags

Help tags are the equivalent, on Mac OS X, of Help balloons on Mac OS 8/9, and appear when the cursor hovers over a user interface element. Carbon also supports Help tags on Mac OS 8/9; however, you may consider that, compared with Help balloons, their "look" is somewhat at odds with the Platinum appearance.

Human Interface Guidelines

Guidelines for Help tags are at http://developer.apple.com/techpubs/macosx/Carbon/pdf/UsingHelpTags.pdf.

Creating Help Tags

You create a Help tag for a control by filling in the fields of an `HMHelpContentRec` structure and its associated `HMHelpContent` structure and passing the control reference and the address of the `HMHelpContentRec` structure in a call to the function `HMSetControlHelpContent`.

Data Types

The an `HMHelpContentRec` and `HMHelpContent` structures are as follows:

```
struct HMHelpContentRec
{
  SInt32           version;
  Rect             absHotRect;
  HMTagDisplaySide tagSide;
  HMHelpContent    content[2];
} ;
typedef struct HMHelpContentRec HMHelpContentRec;
typedef HMHelpContentRec *HMHelpContentPtr;

struct HMHelpContent
{
  HMContentType  contentType;
  union
  {
    CFStringRef     tagCFString;   // A CFStringRef reference.
    Str255          tagString;     // A Pascal string.
    HMStringResType tagStringRes;  // A 'STR#' resource ID and index.
    TEHandle        tagTEHandle;   // A TextEdit handle. (Mac OS 8/9 only.)
    SInt16          tagTextRes;    // A 'TEXT'/'styl' resource ID. (Mac OS 8/9 only.)
    SInt16          tagStrRes;     // A 'STR ' resource ID
  }                  u;
} ;
typedef struct HMHelpContent HMHelpContent;
```

The `HMStringResType` structure is used to specify the resource ID and index when Help tag content is sourced from a `'STR#'` resource:

```
struct HMStringResType
{
  short hmmResID; // Resource ID of 'STR#' resource.
  short hmmIndex; // 'STR#' resource index.
} ;
typedef struct HMStringResType HMStringResType;
```

Note that the `content` field of the `HMHelpContentRec` structure is a two-element array. The second element allows you to provide an expanded version of the Help tag message when the user presses the Command key.

Constants

Typical constants relevant to the `tagSide` field of the `HMHelpContentRec` structure are as follows:

Constant	*Value*	*Description*
`kHMDefaultSide`	`0`	System default location.
`kHMOutsideTopScriptAligned`	`1`	Above, aligned with left or right depending on system script.
`kHMOutsideLeftCenterAligned`	`2`	To the left, centered vertically.
`kHMOutsideRightCenterAligned`	`4`	To the right, centered vertically.
`kHMOutsideTopLeftAligned`	`5`	Above, aligned with left.
`kHMOutsideTopRightAligned`	`6`	Above, aligned with right.
`kHMOutsideLeftTopAligned`	`7`	To the left, aligned with top.
`kHMOutsideLeftBottomAligned`	`8`	To the left, aligned with bottom.
`kHMOutsideBottomLeftAligned`	`9`	Below, aligned with left.
`kHMOutsideBottomRightAligned`	`10`	Below, aligned with right.

Constants relevant to the `content` field of the `HMHelpContentRec` structure are as follows:

Constant	*Value*	*Description*
`kHMMinimumContentIndex`	`0`	First element of content array.
`kHMMaximumContentIndex`	`1`	Second element of content array.

Constants relevant to the `contentType` field of the `HMHelpContent` structure are as follows:

Constant	*Value*	*Description*
`kHMNoContent`	`'none'`	No content.
`kHMCFStringContent`	`'cfst'`	Content sourced from a `CFStringRef` reference.
`kHMPascalStrContent`	`'pstr'`	Content sourced from a Pascal string.
`kHMStringResContent`	`'str#'`	Content sourced from a `'STR#'` resource.
`kHMTEHandleContent`	`'txth'`	Content sourced from a TexEdit handle. (Mac OS 8/9 only.)
`kHMTextResContent`	`'text'`	Content sourced from `'TEXT'`/`'styl'` resources. (Mac OS 8/9 only.)
`kHMStrResContent`	`'str '`	Content sourced from a `'STR '` resource.

Help Tags for Windows

You create a Help tag for a window in the same way you do for a control, except that

- You call the function `HMSetWindowHelpContent` instead of `HMSetControlHelpContent`.
- You assign the window's port rectangle, converted to global coordinates, to the `absHotRect` field of the `HMHelpContentRec` structure.
- You must call `HMSetControlHelpContent` again whenever the window size or position changes to ensure that

the hot rectangle coordinates are updated.

Help Tags for Menus

You create Help tags for menu titles and items in the same way as you do for a control, except that you call the function `HMSetMenuItemHelpContent` instead of `HMSetControlHelpContent`.

At the time of writing, menu title and menu item Help tags were not supported by Carbon or CarbonLib. However, future support was planned.

Setting the Delay Before Tag Display

You can set the delay, in milliseconds, before a tag opens by calling the function `HMSetTagDelay`.

Enabling and Disabling Help Tags

You can enable and disable help tags using the function `HMSetHelpTagsDisplayed`, and you can determine whether Help tags are currently enabled using the function `HMAreHelpTagsDisplayed`.

Ensuring Compatibility with the Operating Environment

If your application is to run successfully in the software and hardware environments that may be present in a wide range of Macintosh models, it must be able to acquire information about a number of machine-dependent features and, where appropriate, act on that information.

The Gestalt Function

The `Gestalt` function may be used to acquire a wide range of information about the operating environment.

```
OSErr  Gestalt(OSType selector,long *response);
```

Returns: Error code. (0 = no error.)

`Selector` Selector code.

`Response` 4-byte return result which provides the requested information. When all four bytes are not needed, the result is expressed in the low-order byte.

The types of information capable of being retrieved by `Gestalt` are as follows:

- The type of machine.
- The version of the System file currently running.
- The type of CPU.
- The type of keyboard attached to the machine.
- The type of floating-point unit (FPU) installed, if any.
- The type of memory management unit (MMU).
- The size of the available RAM.
- The amount of available virtual memory.
- The versions and features of various drivers and managers.

Gestalt Selectors

To use `Gestalt`, you pass it a selector, which specifies exactly what information your application is seeking. Of those selectors that are pre-defined by the Gestalt Manager, there are two sub-types:

- **Environmental Selectors.** Environmental selectors are those that return information about the existence, or otherwise, of a feature. This information can be used by your application to guide its actions. Some examples of the many available environmental selectors, and the information returned in the `response` parameter, are as follows:

Selector	*Information Returned*
`gestaltFPUType`	FPU type.
`gestaltKeyboardType`	Keyboard type.
`gestaltPhysicalRAMSize`	Physical RAM size.
`gestaltQuickdrawVersion`	QuickDraw version.
gestaltTextEditVersion	TextEdit version.

- **Informational Selectors.** Informational selectors are those that provide information that should be used for the user's enlightenment only. This information should never be used as proof positive of some feature's existence, nor should it be used to guide your application's actions. `gestaltMachineType` is an example of an informational selector.

Gestalt Responses

In most cases, the last few characters in the selector's name form a suffix that indicates the type of value that will be returned in the `response` parameter. The following shows the meaningful suffixes:

Suffix	*Returned Value*
`Attr`	A range of 32 bits, the meaning of which must be determined by comparison with a list of constants.
`Count`	A number indicating how many of the indicated type of items exist.
`Size`	A size, usually in bytes.
`Table`	Base address of a table.
`Type`	An index describing a particular type of feature.
`Version`	A version number. Implied decimal points may separate digits of the returned value. For example, a value of `0x0910` returned in response to the `gestaltSystemVersion` selector means that system software version 9.1.0 is present.

Using Gestalt

The header file Gestalt.h defines and describes Gestalt Manager selectors, together with the many constants, which may be used to test the `response` parameter.

Example 1

For example, when `Gestalt` is used to check whether Version 1.3 or later of Color QuickDraw is present, the value returned in the `response` parameter may be compared with `gestalt32BitQD13` as follows:

```
OSErr   osErr
SInt32  response;
Boolean colorQuickDrawVers13Present = true;

osErr = Gestalt(gestaltQuickdrawVersion,&response);
if(osErr == noErr)
{
  if(response < gestalt32BitQD13)
    colorQuickDrawVers13Present = false;
}
```

Example 2

Many constants in Gestalt.h represent bit numbers. In this example, the value returned in the `response` parameter is tested to determine whether bit number 5 (`gestaltHasSoundInputDevice`) is set:

```
OSErr   osErr;
SInt32  response;
Boolean hasSoundInputDevice = false;

osErr = Gestalt(gestaltSoundAttr,&response);
if(osErr == noErr)
  gHasSoundInputDevice = BitTst(&response,31 - gestaltHasSoundInputDevice);
```

Note that the function `BitTst` is used to determine whether the specified bit is set. Bit numbering with `BitTst` is the opposite of the usual MC680x0 numbering scheme used by `Gestalt`. Thus the bit to be tested must be subtracted from 31. This is illustrated in the following:

```
Bit numbering as used in BitTst
0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
30 31
Bit as numbered in MC69000 CPU operations, and used by Gestalt
15 14 13 12 11 10 9  8  7  6  5  4  3  2  1  0  15 14 13 12 11 10 9  8  7  6  5  4  3  2
1  0
```

```
gestaltHasSoundInputDevice = 5
31 - 5 = 26
```

Carbon and Available APIs

CarbonLib and the Underlying Mac OS 8/9 System Software

CarbonLib (the implementation of the Carbon API for Mac OS 8/9) does not, in general, implement new APIs on old systems. It is basically a "pass-through" library that re-exports what is available on the underlying system software.

Thus, if your application calls a function introduced with Mac OS 9.0, it will compile correctly because that function is, by definition, part of the Carbon API (all new function introduced from Mac OS 8.5 onwards are supported by Carbon); however, if the application is run on Mac OS 8.6, the call to that function will fail.

There are a few exceptions to this rule, for example, the menu, control, and window property APIs.

Interpreting Universal Headers Comments

When you run a CFM application on Mac OS X, CFM follows the same rules for runtime linkages as it does on Mac OS 8/9. Since the application links against a library named CarbonLib, there needs to be a CFM library named CarbonLib somewhere on Mac OS X.

That library is /System/Library/CFM Support/CarbonLib. It exports all of the APIs as does CarbonLib on Mac OS 8/9, plus some APIs that are only implemented on Mac OS X. (Those exports, incidentally, are not the real implementations but simply glue that transfers control from the CFM library to the Mach-O frameworks, where the APIs are really implemented. Frameworks are like Mac OS 8/9 shared libraries.)

In the Universal Headers, a comment like:

```
CarbonLib:  in CarbonLib 1.1 and later
```

means that the API is implemented in CarbonLib for Mac OS 8/9 and exported from the CarbonLib library on Mac OS X. This comment

```
CarbonLib:  in CarbonLib on Mac OS X
```

means that the API is not implemented in CarbonLib for Mac OS 8/9 but is still exported from the CarbonLib library on Mac OS X. The real determinant of whether an API is available in Carbon.framework is this comment:

```
Mac OS X:   in version 10.0 or later
```

If the comment `in CarbonLib on Mac OS X` applies to a function called in your application, you must add the stub library CarbonFrameworkLib to the CodeWarrior project.

T-Vector Tests

In Carbon, the recommended method for checking the availability of a routine is to check its T-Vector

(transition vector) directly, as in the following example.

```
if((UInt32) CreateStandardSheet == (UInt32) kUnresolvedCFragSymbolAddress)
  // CreateStandardSheet is not available. Do this.
else
  // Do this.
```

Main Notification Manager Data Types and Functions

Data Types

Notification Structure

```
struct NMRec
{
  QElemPtr  qLink;       // Next queue entry.
  short     qType;       // Queue type.
  short     nmFlags;     // (Reserved.)
  long      nmPrivate;   // (Reserved.)
  short     nmReserved;  // (Reserved.)
  short     nmMark;      // Item to mark in Apple menu.
  Handle    nmIcon;      // Handle to small icon.
  Handle    nmSound;     // Handle to sound structure.
  StringPtr nmStr;       // String to appear in notification.
  NMUPP     nmResp;      // Pointer to response function.
  long      nmRefCon;    // For application use.
} ;
typedef struct NMRec NMRec;
typedef NMRec *NMRecPtr;
```

Functions

Add Notification Request to the Notification Queue

```
OSErr  NMInstall(NMRecPtr nmReqPtr);
```

Remove Notification Request from the Notification Queue

```
OSErr  NMRemove(NMRecPtr nmReqPtr);
```

Relevant Process Manager Data Types and Functions

Data Types

Process Serial Number

```
struct ProcessSerialNumber
{
  unsigned long  highLongOfPSN;
  unsigned long  lowLongOfPSN;
} ;
```

Functions

Getting Process Serial Numbers

```
OSErr  GetCurrentProcess(ProcessSerialNumber *PSN);
OSErr  GetFrontProcess(ProcessSerialNumber *PSN);
```

Comparing Two Process Serial Numbers

```
OSErr  SameProcess(const ProcessSerialNumber *PSN1,const ProcessSerialNumber *PSN2,
       Boolean *result);
```

Relevant Event Manager Function

Check For Command-Period

```
Boolean  CheckEventQueueForUserCancel(void);
```

Relevant Color Picker Utilities Function

```
Boolean  GetColor(Point where,ConstStr255Param prompt,const RGBColor *inColor,
         RGBColor *outColor);
```

Relevant QuickDraw Function

Drawing Across Multiple Video Devices

```
void      DeviceLoop(RgnHandle drawingRgn,DeviceLoopDrawingUP drawingProc,long userData,
          DeviceLoopFlags flags);
```

Relevant Window Manager Function

```
OSStatus  ConstrainWindowToScreen(WindowRef inWindowRef,WindowRegionCode inRegionCode,
                                  WindowConstrainOptions inOptions,const Rect *inScreenRect,
                                  Rect *outStructure);
```

Main Help Package Constants, Data Types, and Functions

Constants

Content Type

```
kHMNoContent                  = FOUR_CHAR_CODE('none')
kHMCFStringContent            = FOUR_CHAR_CODE('cfst')
kHMPascalStrContent           = FOUR_CHAR_CODE('pstr')
kHMStringResContent           = FOUR_CHAR_CODE('str#')
kHMTEHandleContent            = FOUR_CHAR_CODE('txth')
kHMTextResContent             = FOUR_CHAR_CODE('text')
kHMStrResContent              = FOUR_CHAR_CODE('str ')
```

Tag Display Side

```
kHMDefaultSide                = 0
kHMOutsideTopScriptAligned    = 1
kHMOutsideLeftCenterAligned   = 2
kHMOutsideBottomScriptAligned = 3
kHMOutsideRightCenterAligned  = 4
kHMOutsideTopLeftAligned      = 5
kHMOutsideTopRightAligned     = 6
kHMOutsideLeftTopAligned      = 7
kHMOutsideLeftBottomAligned   = 8
kHMOutsideBottomLeftAligned   = 9
kHMOutsideBottomRightAligned  = 10
kHMOutsideRightTopAligned     = 11
kHMOutsideRightBottomAligned  = 12
kHMOutsideTopCenterAligned    = 13
kHMOutsideBottomCenterAligned = 14
kHMInsideRightCenterAligned   = 15
kHMInsideLeftCenterAligned    = 16
kHMInsideBottomCenterAligned  = 17
kHMInsideTopCenterAligned     = 18
kHMInsideTopLeftCorner        = 19
kHMInsideTopRightCorner       = 20
kHMInsideBottomLeftCorner     = 21
kHMInsideBottomRightCorner    = 22
kHMAbsoluteCenterAligned      = 23
```

For HMHelpContentRec.content

```
kHMMinimumContentIndex        = 0
kHMMaximumContentIndex        = 1
```

Data Types

```
typedef UInt32 HMContentType;
typedef SInt16 HMTagDisplaySide;
```

HelpContent

```
struct HMHelpContent
{
  HMContentType  contentType;
  union
  {
```

```
    CFStringRef     tagCFString;
    Str255          tagString;
    HMStringResType tagStringRes;
    TEHandle        tagTEHandle;
    SInt16          tagTextRes;
    SInt16          tagStrRes;
  }                  u;
} ;
typedef struct HMHelpContent  HMHelpContent;
```

HMHelpContentRec

```
struct HMHelpContentRec
{
  SInt32           version;
  Rect             absHotRect;
  HMTagDisplaySide tagSide;
  HMHelpContent    content[2];
} ;
typedef struct HMHelpContentRec HMHelpContentRec;
typedef HMHelpContentRec *HMHelpContentPtr;
```

HMStringResType

```
struct HMStringResType
{
  short hmmResID;
  short hmmIndex;
} ;
typedef struct HMStringResType HMStringResType;
```

Functions

Installing and Retrieving Content

```
OSStatus  HMSetControlHelpContent(ControlRef inControl,const HMHelpContentRec *inContent);
OSStatus  HMGetControlHelpContent(ControlRef inControl,HMHelpContentRec *outContent);
OSStatus  HMSetWindowHelpContent(WindowRef inWindow,const HMHelpContentRec *inContent);
OSStatus  HMGetWindowHelpContent(WindowRef inWindow,HMHelpContentRec *outContent);
OSStatus  HMSetMenuItemHelpContent(MenuRef inMenu,MenuItemIndex inItem,
          const HMHelpContentRec *inContent);
OSStatus  HMGetMenuItemHelpContent(MenuRef inMenu,MenuItemIndex inItem,
          HMHelpContentRec *outContent);
```

Enabling and Disabling Help Tags

```
Boolean   HMAreHelpTagsDisplayed(void);
OSStatus  HMSetHelpTagsDisplayed(Boolean inDisplayTags);
```

Setting and Getting Tag Delay

```
OSStatus  HMSetTagDelay(Duration inDelay);
OSStatus  HMGetTagDelay(Duration *outDelay);
```

Displaying Tags

```
OSStatus  HMDisplayTag(const HMHelpContentRec *inContent);
```

Relevant Gestalt Manager Function

```
OSErr     Gestalt(OSType selector,long *response);
```

Demonstration Program Miscellany Listing

```
//
**********************************************************************************************
// Miscellany.h                                                                CARBON EVENT
MODEL
//
**********************************************************************************************
//
// This program demonstrates:
//
// •  The use of the Notification Manager to allow an application running in the background
to
//    to communicate with the foreground application.
//
// •  The use of the determinate progress bar control to show progress during a time-
//    consuming operation, together with scanning the event queue for Command-period key-
down
//    events for the purpose of terminating the lengthy operation  before it concludes of
its
//    own accord.
//
// •  The use of the Color Picker to solicit a choice of color from the user.
//
// •  Image drawing optimisation in a multi-monitors environment.
//
// •  Help tags for controls and windows with minimum and maximum content.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  An 'MBAR' resource, and 'MENU' resources for Apple, File, Edit and Demonstration menus
//    (preload, non-purgeable).
//
// •  A 'DLOG' resource (purgeable), and associated 'DITL', 'dlgx', and 'dftb' resources
//    (purgeable), for a dialog box in which the progress indicator is displayed.
//
// •  'CNTL' resources (purgeable) for the progress indicator dialog.
//
// •  'icn#', 'ics4', and 'ics8' resources (non-purgeable) which contain the application
icon
//    shown in the Application menu during the Notification Manager demonstration.
//
// •  A 'snd ' resource (non-purgeable) used in the Notification Manager demonstration.
//
// •  A 'STR ' resource (non-purgeable) containing the text displayed in the alert box
invoked
//    by the Notification Manager.
//
// •  'STR#' resources (purgeable) containing the label and narrative strings for the
//    notification-related alert displayed by Miscellany and the minimum and maximum Help
tag
//    content.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
//
**********************************************************************************************

// ..........................................................................
```

```
includes

#include <Carbon.h>

// ..........................................................................
defines

#define rMenubar            128
#define mAppleApplication   128
#define  iAbout             1
#define mFile               129
#define  iQuit              12
#define mDemonstration      131
#define  iNotification      1
#define  iProgress          2
#define  iColorPicker      3
#define  iMultiMonitors     4
#define  iHelpTag           5
#define rWindow             128
#define rDialog             128
#define  iProgressIndicator 1
#define rIconFamily         128
#define rBarkSound          8192
#define rString             128
#define rAlertStrings       128
#define  indexLabel         1
#define  indexNarrative     2
#define rPicture            128
#define topLeft(r)          (((Point *) &(r))[0])
#define botRight(r)         (((Point *) &(r))[1])

//.......................................................................... function
prototypes

void      main                          (void);
void      doPreliminaries               (void);
OSStatus  appEventHandler               (EventHandlerCallRef,EventRef,void *);
OSStatus  windowEventHandler            (EventHandlerCallRef,EventRef,void *);
void      doMenuChoice                  (MenuID,MenuItemIndex);
void      doSetUpNotification           (void);
void      doPrepareNotificationStructure (void);
void      doIdle                        (void);
void      doDisplayMessageToUser        (void);

void      doProgressBar                 (void);

void      deviceLoopDraw                (SInt16,SInt16,GDHandle,SInt32);

void      doColorPicker                (void);
void      doDrawColorPickerChoice      (void);
char      *doDecimalToHexadecimal       (UInt16 n);

void      doHelpTagControl              (void);
void      doHelpTagWindow               (void);

//
********************************************************************************
// Miscellany.c
//
********************************************************************************

#include "Miscellany.h"

// ..................................................................... global
variables
```

```
DeviceLoopDrawingUPP gDeviceLoopDrawUPP;
WindowRef            gWindowRef;
ControlRef           gBevelButtonControlRef;
ProcessSerialNumber  gProcessSerNum;
Boolean              gMultiMonitorsDrawDemo = false;
Boolean              gColorPickerDemo  = false;
Boolean              gHelpTagsDemo = false;
RGBColor             gWhiteColor = {  0xFFFF, 0xFFFF, 0xFFFF } ;
RGBColor             gBlueColor  = {  0x6666, 0x6666, 0x9999 } ;

extern Boolean       gNotificationInQueue;

// ********************************************************************************
main

void  main(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  Rect          contentRect = {  100,100,402,545 } ;
  Rect          portRect;
  Rect          controlRect = {  65,10,155,100 } ;
  EventTypeSpec applicationEvents[] = {  {  kEventClassApplication,
kEventAppActivated    } ,
                                       {  kEventClassCommand,
kEventProcessCommand  }  } ;
  EventTypeSpec windowEvents[]      = {  {  kEventClassWindow,
kEventWindowDrawContent    } ,
                                       {  kEventClassWindow, kEventWindowGetIdealSize   } ,
                                       {  kEventClassWindow, kEventWindowGetMinimumSize } ,
                                       {  kEventClassWindow,
kEventWindowBoundsChanged  }  } ;
  // ..................................................................... do
preliminaries

  doPreliminaries();

  // ..................................................... create universal procedure
pointer

  gDeviceLoopDrawUPP = NewDeviceLoopDrawingUPP((DeviceLoopDrawingProcPtr) deviceLoopDraw);

  // ............................................................. set up menu bar and
menus

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    ExitToShell();
  SetMenuBar(menubarHdl);
  DrawMenuBar();

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
      DisableMenuItem(menuRef,0);
    }
  }
  else
  {
    menuRef = GetMenuRef(mFile);
```

```
    if(menuRef != NULL)
      SetMenuItemCommandID(menuRef,iQuit,kHICommandQuit);
  }

  // ........................................... install application event handler and
timer

  InstallApplicationEventHandler(NewEventHandlerUPP((EventHandlerProcPtr) appEventHandler),
                                 GetEventTypeCount(applicationEvents),applicationEvents,
                                 0,NULL);

  InstallEventLoopTimer(GetCurrentEventLoop(),0,1,
                        NewEventLoopTimerUPP((EventLoopTimerProcPtr) doIdle),NULL,NULL);

..................................................................................... open
window

  CreateNewWindow(kDocumentWindowClass,kWindowStandardHandlerAttribute |
                  kWindowStandardDocumentAttributes,&contentRect,&gWindowRef);

  ChangeWindowAttributes(gWindowRef,0,kWindowCloseBoxAttribute);
  SetWTitle(gWindowRef,"\pMiscellany");
  RepositionWindow(gWindowRef,NULL,kWindowAlertPositionOnMainScreen);

  SetPortWindowPort(gWindowRef);
  TextSize(10);
  ShowWindow(gWindowRef);
  GetWindowPortBounds(gWindowRef,&portRect);
  InvalWindowRect(gWindowRef,&portRect);

  // ............................................................ install window event
handler

  InstallWindowEventHandler(gWindowRef,
                            NewEventHandlerUPP((EventHandlerProcPtr) windowEventHandler),
                            GetEventTypeCount(windowEvents),windowEvents,0,NULL);

  // ............................................................ create control and
help tags

  CreateBevelButtonControl(gWindowRef,&controlRect,CFSTR("Control"),
                           kControlBevelButtonNormalBevel,kControlBehaviorPushbutton,
                           NULL,0,0,0,&gBevelButtonControlRef);
  doHelpTagControl();
  HideControl(gBevelButtonControlRef);
  doHelpTagWindow();
  HMSetHelpTagsDisplayed(false);

  // ................................................... get process serial number of this
process

  GetCurrentProcess(&gProcessSerNum);

  // ............................................................... run application
event loop

  RunApplicationEventLoop();
}

// ************************************************************************
doPreliminaries

void  doPreliminaries(void)
{
  MoreMasterPointers(640);
  InitCursor();
```

```
}

// ************************************************************************
appEventHandler

OSStatus  appEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                          void * userData)
{
  OSStatus      result = eventNotHandledErr;
  UInt32        eventClass;
  UInt32        eventKind;
  HICommand     hiCommand;
  MenuID        menuID;
  MenuItemIndex menuItem;

  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);
  switch(eventClass)
  {
    case kEventClassApplication:
      if(eventKind == kEventAppActivated)
      {
        if(gNotificationInQueue)
          doDisplayMessageToUser();
        result = noErr;
      }
      break;

    case kEventClassCommand:
      if(eventKind == kEventProcessCommand)
      {
        GetEventParameter(eventRef,kEventParamDirectObject,typeHICommand,NULL,
                          sizeof(HICommand),NULL,&hiCommand);
        menuID = GetMenuID(hiCommand.menu.menuRef);
        menuItem = hiCommand.menu.menuItemIndex;
        if((hiCommand.commandID != kHICommandQuit) &&
           (menuID >= mAppleApplication && menuID <= mDemonstration))
        {
          doMenuChoice(menuID,menuItem);
          result = noErr;
        }
      }
      break;
  }

  return result;
}

// ************************************************************************
windowEventHandler

OSStatus  windowEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                             void* userData)
{
  OSStatus  result = eventNotHandledErr;
  UInt32    eventClass;
  UInt32    eventKind;
  WindowRef windowRef;
  SInt32    deviceLoopUserData;
  RgnHandle regionHdl;
  Rect      portRect, positioningBounds;
  Point     idealHeightAndWidth, minimumHeightAndWidth;

  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);
```

```
  switch(eventClass)
  {
    case kEventClassWindow:
      GetEventParameter(eventRef,kEventParamDirectObject,typeWindowRef,NULL,sizeof
(windowRef),
                        NULL,&windowRef);
      switch(eventKind)
      {
        case kEventWindowDrawContent:
          if(gMultiMonitorsDrawDemo)
          {
            RGBBackColor(&gWhiteColor);
            deviceLoopUserData = (SInt32) windowRef;
            regionHdl = NewRgn();
            if(regionHdl)
            {
              GetPortVisibleRegion(GetWindowPort(windowRef),regionHdl);
              DeviceLoop(regionHdl,gDeviceLoopDrawUPP,deviceLoopUserData,0);
              DisposeRgn(regionHdl);
            }
          }
          else if(gColorPickerDemo )
          {
            RGBBackColor(&gBlueColor);
            GetWindowPortBounds(windowRef,&portRect);
            EraseRect(&portRect);
            doDrawColorPickerChoice();
          }
          else
          {
            RGBBackColor(&gBlueColor);
            GetWindowPortBounds(windowRef,&portRect);
            EraseRect(&portRect);
            if(gHelpTagsDemo)
            {
              Draw1Control(gBevelButtonControlRef);
              RGBForeColor(&gWhiteColor);
              MoveTo(10,20);
              DrawString("\pHover the cursor in the window, and over the bevel button, ");
              DrawString("\puntil the Help tag appears.");
              MoveTo(10,35);
              DrawString("\pPress the Command key to invoke the maximum content.");
              MoveTo(10,50);
              DrawString("\pDrag the window to a new location.");
            }
          }
          result = noErr;
          break;

        case kEventWindowGetIdealSize:
          GetAvailableWindowPositioningBounds(GetMainDevice(),&positioningBounds);
          idealHeightAndWidth.v = positioningBounds.bottom;
          idealHeightAndWidth.h = positioningBounds.right;
          SetEventParameter(eventRef,kEventParamDimensions,typeQDPoint,
                            sizeof(idealHeightAndWidth),&idealHeightAndWidth);
           result = noErr;
          break;
        case kEventWindowGetMinimumSize:
          minimumHeightAndWidth.v = 302;
          minimumHeightAndWidth.h = 445;
          SetEventParameter(eventRef,kEventParamDimensions,typeQDPoint,
                            sizeof(minimumHeightAndWidth),&minimumHeightAndWidth);
          result = noErr;
          break;

        case kEventWindowBoundsChanged:
```

```
          doHelpTagWindow();
          GetWindowPortBounds(windowRef,&portRect);
          InvalWindowRect(windowRef,&portRect);
          result = noErr;
          break;
      }
      break;
  }

  return result;
}

// ********************************************************************************
doMenuChoice

void  doMenuChoice(MenuID menuID,MenuItemIndex menuItem)
{
  Rect portRect;

  if(menuID == 0)
    return;

  switch(menuID)
  {
    case mAppleApplication:
      if(menuItem == iAbout)
        SysBeep(10);
      break;

    case mDemonstration:
      gMultiMonitorsDrawDemo = gColorPickerDemo = gHelpTagsDemo = false;
      if(HMAreHelpTagsDisplayed)
        HMSetHelpTagsDisplayed(false);
      HideControl(gBevelButtonControlRef);
      GetWindowPortBounds(gWindowRef,&portRect);

      switch(menuItem)
      {
        case iNotification:
          RGBBackColor(&gBlueColor);
          EraseRect(&portRect);
          doSetUpNotification();
          break;
        case iProgress:
          RGBBackColor(&gBlueColor);
          EraseRect(&portRect);
          doProgressBar();
          break;

        case iColorPicker:
          gColorPickerDemo = true;
          doColorPicker();
          break;

        case iMultiMonitors:
          gMultiMonitorsDrawDemo = true;
          InvalWindowRect(gWindowRef,&portRect);
          break;

        case iHelpTag:
          gHelpTagsDemo = true;
          InvalWindowRect(gWindowRef,&portRect);
          ShowControl(gBevelButtonControlRef);
          HMSetHelpTagsDisplayed(true);
          break;
      }
```

```
      break;
  }
}

//
**********************************************************************************************
// Notification.c
//
**********************************************************************************************

#include "Miscellany.h"

// ..................................................................... global
variables

NMRec                     gNotificationStructure;
long                      gStartingTickCount;
Boolean                   gNotificationDemoInvoked;
Boolean                   gNotificationInQueue;
extern WindowRef          gWindowRef;
extern ProcessSerialNumber gProcessSerNum;
extern RGBColor           gWhiteColor;
extern RGBColor           gBlueColor;

// ********************************************************************
doSetUpNotification

void  doSetUpNotification(void)
{

  doPrepareNotificationStructure();
  gNotificationDemoInvoked = true;

  gStartingTickCount = TickCount();
  RGBForeColor(&gWhiteColor);
  MoveTo(10,279);
  DrawString("\pPlease click on the desktop now to make the Finder ");
  DrawString("\pthe frontmost application.");
  MoveTo(10,292);
  DrawString("\p(This application will post a notification 10 seconds from now.)");
}

// ************************************************************
doPrepareNotificationStructure

void  doPrepareNotificationStructure(void)
{
  Handle       iconSuiteHdl;
  Handle       soundHdl;
  StringHandle stringHdl;

  GetIconSuite(&iconSuiteHdl,rIconFamily,kSelectorAllSmallData);
  soundHdl = GetResource('snd ',rBarkSound);
  stringHdl = GetString(rString);

  gNotificationStructure.qType    = nmType;
  gNotificationStructure.nmMark   = 1;
  gNotificationStructure.nmIcon   = iconSuiteHdl;
  gNotificationStructure.nmSound  = soundHdl;
  gNotificationStructure.nmStr    = *stringHdl;
  gNotificationStructure.nmResp   = NULL;
  gNotificationStructure.nmRefCon = 0;
}

// ****************************************************************************
```

doIdle

```
void  doIdle(void)
{
  ProcessSerialNumber frontProcessSerNum;
  Boolean             isSameProcess;
  Rect                portRect;

  if(gNotificationDemoInvoked)
  {
    if(TickCount() > gStartingTickCount + 600)
    {
      GetFrontProcess(&frontProcessSerNum);
      SameProcess(&frontProcessSerNum,&gProcessSerNum,&isSameProcess);

      if(!isSameProcess)
      {
        NMInstall(&gNotificationStructure);
        gNotificationDemoInvoked = false;
        gNotificationInQueue = true;
      }
      else
      {
        doDisplayMessageToUser();
        gNotificationDemoInvoked = false;
      }
      GetWindowPortBounds(gWindowRef,&portRect);
      EraseRect(&portRect);
    }
  }
}

// ************************************************************************
doDisplayMessageToUser

void  doDisplayMessageToUser(void)
{
  Rect                  portRect;
  AlertStdAlertParamRec paramRec;
  Str255                labelText;
  Str255                narrativeText;
  SInt16                itemHit;

  if(gNotificationInQueue)
  {
    NMRemove(&gNotificationStructure);
    gNotificationInQueue = false;
  }

  GetWindowPortBounds(gWindowRef,&portRect);
  EraseRect(&portRect);

  paramRec.movable       = true;
  paramRec.helpButton    = false;
  paramRec.filterProc    = NULL;
  paramRec.defaultText   = (StringPtr) kAlertDefaultOKText;
  paramRec.cancelText    = NULL;
  paramRec.otherText     = NULL;
  paramRec.defaultButton = kAlertStdAlertOKButton;
  paramRec.cancelButton  = 0;
  paramRec.position      = kWindowDefaultPosition;

  GetIndString(labelText,rAlertStrings,indexLabel);
  GetIndString(narrativeText,rAlertStrings,indexNarrative);

  StandardAlert(kAlertNoteAlert,labelText,narrativeText,&paramRec,&itemHit);
```

```
  DisposeIconSuite(gNotificationStructure.nmIcon,false);
  ReleaseResource(gNotificationStructure.nmSound);
  ReleaseResource((Handle) gNotificationStructure.nmStr);
}

//
**********************************************************************************************
// ProgressIndicator.c
//
**********************************************************************************************

#include "Miscellany.h"

// ..................................................................... global
variables

extern WindowRef gWindowRef;
extern RGBColor  gWhiteColor;
// ************************************************************************
doProgressBar

void  doProgressBar(void)
{
  DialogRef  dialogRef;
  RgnHandle  visRegionHdl = NewRgn();
  ControlRef progressBarRef;
  SInt16     statusMax, statusCurrent;
  SInt16     a, b, c;
  Handle     soundHdl;
  Rect       portRect, theRect;
  RGBColor   redColor = {  0xFFFF, 0x0000, 0x0000 } ;

  if(!(dialogRef = GetNewDialog(rDialog,NULL,(WindowRef) -1)))
    ExitToShell();

  SetPortDialogPort(dialogRef);
  GetPortVisibleRegion(GetWindowPort(GetDialogWindow(dialogRef)),visRegionHdl);
  UpdateControls(GetDialogWindow(dialogRef),visRegionHdl);
  QDFlushPortBuffer(GetDialogPort(dialogRef),NULL);

  SetPortWindowPort(gWindowRef);
  GetWindowPortBounds(gWindowRef,&portRect);

  GetDialogItemAsControl(dialogRef,iProgressIndicator,&progressBarRef);

  statusMax = 3456;
  statusCurrent = 0;
  SetControlMaximum(progressBarRef,statusMax);

  for(a=0;a<9;a++)
  {
    for(b=8;b<423;b+=18)
    {
      for(c=8;c<286;c+=18)
      {
        if(CheckEventQueueForUserCancel())
        {
          soundHdl = GetResource('snd ',rBarkSound);
          SndPlay(NULL,(SndListHandle) soundHdl,false);
          ReleaseResource(soundHdl);
          DisposeDialog(dialogRef);

          EraseRect(&portRect);
          MoveTo(10,292);
          RGBForeColor(&gWhiteColor);
```

```
          DrawString("\pOperation cancelled at user request");

          return;
        }

        SetRect(&theRect,b+a,c+a,b+17-a,c+17-a);
        if(a < 3)                RGBForeColor(&gWhiteColor);
        else if(a > 2 && a < 6)  RGBForeColor(&redColor);
        else if(a > 5)           RGBForeColor(&gWhiteColor);
        FrameRect(&theRect);
        QDFlushPortBuffer(GetWindowPort(gWindowRef),NULL);
        QDFlushPortBuffer(GetDialogPort(dialogRef),NULL);

        SetControlValue(progressBarRef,statusCurrent++);
      }
    }
  }

  DisposeRgn(visRegionHdl);
  DisposeDialog(dialogRef);
  EraseRect(&portRect);
  MoveTo(10,292);
  RGBForeColor(&gWhiteColor);
  DrawString("\pOperation completed");
}

//
**********************************************************************************************
// ColorPicker.c
//
**********************************************************************************************

#include "Miscellany.h"

// ..................................................................... global
variables

RGBColor         gInColor = {  0xCCCC, 0x0000, 0x0000 } ;
RGBColor         gOutColor;
Boolean          gColorPickerButton;
extern WindowRef gWindowRef;
extern RGBColor  gWhiteColor;
extern RGBColor  gBlueColor;

// ************************************************************************
doColorPicker

void  doColorPicker(void)
{
  Rect   portRect, theRect;
  Point  where;
  Str255 prompt = "\pChoose a rectangle color:";

  GetWindowPortBounds(gWindowRef,&portRect);
  theRect = portRect;

  RGBBackColor(&gBlueColor);
  EraseRect(&theRect);
  InsetRect(&theRect,55,55);
  RGBForeColor(&gInColor);
  PaintRect(&theRect);

  where.v = where.h = 0;

  gColorPickerButton = GetColor(where,prompt,&gInColor,&gOutColor);
  InvalWindowRect(gWindowRef,&portRect);
```

```
}

// ************************************************************************
doDrawColorPickerChoice

void  doDrawColorPickerChoice(void)
{
  Rect portRect;
  char *cString;

  GetWindowPortBounds(gWindowRef,&portRect);
  InsetRect(&portRect,55,55);

  if(gColorPickerButton)
  {
    RGBForeColor(&gOutColor);
    PaintRect(&portRect);

    RGBForeColor(&gWhiteColor);

    MoveTo(55,22);
    DrawString("\pRequested Red Value: ");
    cString = doDecimalToHexadecimal(gOutColor.red);
    MoveTo(170,22);
    DrawText(cString,0,6);

    MoveTo(55,35);
    DrawString("\pRequested Green Value: ");
    cString = doDecimalToHexadecimal(gOutColor.green);
    MoveTo(170,35);
    DrawText(cString,0,6);

    MoveTo(55,48);
    DrawString("\pRequested Blue Value: ");
    cString = doDecimalToHexadecimal(gOutColor.blue);
    MoveTo(170,48);
    DrawText(cString,0,6);
  }
  else
  {
    RGBForeColor(&gInColor);
    PaintRect(&portRect);

    RGBForeColor(&gWhiteColor);
    MoveTo(55,48);
    DrawString("\pCancel button was clicked.");
  }
}
// ************************************************************************
doDecimalToHexadecimal

char  *doDecimalToHexadecimal(UInt16 decimalNumber)
{
  static char cString[] = "0xXXXX";
  char        *hexCharas = "0123456789ABCDEF";
  SInt16      a;

  for (a=0;a<4;decimalNumber >>= 4,++a)
    cString[5 - a] = hexCharas[decimalNumber & 0xF];

  return cString;
}

//
****************************************************************************************************
// MultiMonitor.c
```

```
//
**********************************************************************************************

#include "Miscellany.h"

// **************************************************************************
deviceLoopDraw

void  deviceLoopDraw(SInt16 depth,SInt16 deviceFlags,GDHandle targetDeviceHdl,SInt32
userData)
{
  RGBColor  oldForeColor;
  WindowRef windowRef;
  Rect      portRect;
  RGBColor  greenColor  = {  0x0000, 0xAAAA, 0x1111 } ;
  RGBColor  redColor    = {  0xAAAA, 0x4444, 0x3333 } ;
  RGBColor  blueColor   = {  0x5555, 0x4444, 0xFFFF } ;
  RGBColor  ltGrayColor = {  0xDDDD, 0xDDDD, 0xDDDD } ;
  RGBColor  grayColor   = {  0x9999, 0x9999, 0x9999 } ;
  RGBColor  dkGrayColor = {  0x4444, 0x4444, 0x4444 } ;

  GetForeColor(&oldForeColor);

  windowRef = (WindowRef) userData;
  GetWindowPortBounds(windowRef,&portRect);
  EraseRect(&portRect);

  if(((1 << gdDevType) & deviceFlags) != 0)
  {
    InsetRect(&portRect,50,50);
    RGBForeColor(&greenColor);
    PaintRect(&portRect);
    InsetRect(&portRect,40,40);
    RGBForeColor(&redColor);
    PaintRect(&portRect);
    InsetRect(&portRect,40,40);
    RGBForeColor(&blueColor);
    PaintRect(&portRect);
  }
  else
  {
    InsetRect(&portRect,50,50);
    RGBForeColor(&ltGrayColor);
    PaintRect(&portRect);
    InsetRect(&portRect,40,40);
    RGBForeColor(&grayColor);
    PaintRect(&portRect);
    InsetRect(&portRect,40,40);
    RGBForeColor(&dkGrayColor);
    PaintRect(&portRect);
  }

  RGBForeColor(&oldForeColor);
}

//
**********************************************************************************************
// HelpTag.c
//
**********************************************************************************************

#include "Miscellany.h"
#include <string.h>

// ..................................................................... global
variables
```

```
extern ControlRef gBevelButtonControlRef;
extern WindowRef  gWindowRef;

// ....................................................................
doHelpTagControl

void  doHelpTagControl(void)
{
  HMHelpContentRec helpContent;

  memset(&helpContent,0,sizeof(helpContent));
  HMSetTagDelay(50);

  helpContent.version = kMacHelpVersion;
  helpContent.tagSide = kHMOutsideBottomLeftAligned;

  helpContent.content[kHMMinimumContentIndex].contentType = kHMStringResContent;
  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmResID = 129;
  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 1;
  helpContent.content[kHMMaximumContentIndex].contentType = kHMStringResContent;
  helpContent.content[kHMMaximumContentIndex].u.tagStringRes.hmmResID = 129;
  helpContent.content[kHMMaximumContentIndex].u.tagStringRes.hmmIndex = 2;

  HMSetControlHelpContent(gBevelButtonControlRef,&helpContent);
}
// ....................................................................
doHelpTagWindow

void  doHelpTagWindow(void)
{
  Rect              hotRect;
  HMHelpContentRec  helpContent;

  memset(&helpContent,0,sizeof(helpContent));
  HMSetTagDelay(500);

  helpContent.version = kMacHelpVersion;
  helpContent.tagSide = kHMOutsideRightCenterAligned;

  GetWindowPortBounds(gWindowRef,&hotRect);
  LocalToGlobal(&topLeft(hotRect));
  LocalToGlobal(&botRight(hotRect));
  helpContent.absHotRect = hotRect;

  helpContent.content[kHMMinimumContentIndex].contentType = kHMStringResContent;
  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmResID = 129;
  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 3;
  helpContent.content[kHMMaximumContentIndex].contentType = kHMStringResContent;
  helpContent.content[kHMMaximumContentIndex].u.tagStringRes.hmmResID = 129;
  helpContent.content[kHMMaximumContentIndex].u.tagStringRes.hmmIndex = 4;

  HMSetWindowHelpContent(gWindowRef,&helpContent);
}

//
********************************************************************************
```

Demonstration Program Miscellany Comments

When this program is run, the user should make choices from the Demonstration menu, taking the following actions and making the following observations:

- Choose the Notification item and, observing the instructions in the window, click the desktop immediately to make the Finder the foreground application. A notification will be posted by Miscellany about 10 seconds after the Notification item choice is made. When about 10 seconds have elapsed, the Notification Manager invokes an alert (Mac OS 8.6), floating window (Mac OS 9.x), or system movable modal alert (Mac OS X) and alternates the Finder and Miscellany icons in the OS 8/9 Application menu title. Observing the instructions in the alert/floating window/system movable modal alert:

    - Dismiss the alert (Mac OS 8.6 only).

    - On Mac OS 8/9, then choose the Miscellany item in the OS 8/9 Application menu, noting the • mark to the left of the item name. When Miscellany comes to the foreground, note that the icon alternation concludes and that an alert (invoked by Miscellany) appears. Dismiss this second alert.

      On Mac OS X, click on the application's icon in the Dock.

- Choose the Notification item again and this time leave Miscellany in the foreground. Note that only the alert invoked by Miscellany appears on this occasion.

- Choose the Notification item again and, this time, click on the desktop and then in the Miscellany window before 10 seconds elapse. Note again that only the alert invoked by Miscellany appears.

- Choose the Determinate Progress Indicator item, noting that the progress indicator dialog is automatically disposed of when the (simulated) time-consuming task concludes.

- Choose the Determinate Progress Indicator item again, and this time press the Command-period key combination before the (simulated) time-consuming task concludes. Note that the progress indicator dialog is disposed of when the Command-period key combination is pressed.

- Choose the Color Picker item and make color choices using the various available modes. Note that, when the Color Picker is dismissed by clicking the OK button,

the requested RGB color values for the chosen color are displayed in hexadecimal, together with a rectangle in that color, in the Miscellany window.

- Choose the Multiple Monitors Draw item, noting that the drawing of the simple demonstration image is optimised as follows:
    - On a monitor set to display 256 or more colors, the image is drawn in three distinct colors. The luminance of the three colors is identical, meaning that, if these colors are drawn on a grayscale screen, they will all appear in the same shade of gray.
    - On a monitor set to display 256 shades of gray, the image is drawn in three distinct shades of gray.
- Choose the Help Tags item, hover the cursor over the window, and when the Help tag appears, press the Command key to observe the maximum content version of the tag. Repeat this while hovering the cursor over the bevel button control.

Miscellany.c

Global Variables

`gDeviceLoopDrawUPP` will be assigned a universal procedure pointer to the image-optimizing drawing function `deviceLoopDraw` called by DeviceLoop. `gProcessSerNum` will be assigned the process serial number of the Miscellany application.

main

The call to `NewDeviceLoopDrawingProc` creates a universal procedure pointer to the image-optimizing drawing function `deviceLoopDraw`.

A timer is installed and set to fire every one second. When it fires, the function `doIdle` is called.

A bevel button control is created, following which the calls to `doHelpTagControl` and `doHelpTagWindow` create Help tags for the bevel button control and the window. `HMSetHelpTagsDisplayed` is called to disable the tags until the Help Tags item is chosen from the Demonstration menu.

`GetCurrentProcess` gets the process serial number of this process. The timer and the process serial number are used in the notification demonstration.

appEventHandler

When the `kEventAppActivated` event type is received, if the global variable `gNotificationInQueue` is set to true, `doDisplayMessageToUser` is called. This is part of the notification demonstration.

windowEventHandler

When the `kEventWindowDrawContent` event type is received, if the Multiple Monitors Draw item in the Demonstration menu has been chosen (`gMultiMonitorsDrawDemo` is true), a call is made to DeviceLoop and the universal procedure pointer to the application-defined (callback) drawing function `deviceLoopDraw` is passed as the second parameter.

doMenuChoice

When the Multiple Monitors Draw item in the Demonstration menu is chosen, the window's port rectangle is invalidated so as to force a `kEventWindowDrawContent` event and consequential call to DeviceLoop.

Notification.c

doSetUpNotification

`doSetUpNotification` is called when the user chooses Notification from the Demonstration menu.

The first line calls `doPrepareNotificationStructure`, which fills in the relevant fields of a notification structure. The next line assigns true to a global variable which records that the Notification item has been chosen by the user.

The next line saves the system tick count at the time that the user chose the Notification item. This value is used later to determine when 10 seconds have elapsed following the execution of this line.

doPrepareNotificationStructure

`doPrepareNotificationStructure` fills in the relevant fields of the notification structure.

First, however, `GetIconSuite` creates an icon suite based on the specified resource ID and the third parameter, which limits the suite to 'ics#', 'ics4' and 'ics8' icons. The `GetIconSuite` call returns the handle to the suite in its first parameter. The call to

`GetResource` loads the specified `'snd '` resource. `GetString` loads the specified `'STR '` resource.

The first line of the main block specifies the type of operating system queue. The next line specifies that the • mark is to appear next to the application's name in the Mac OS 8/9 Application menu. The next three lines assign the icon suite (for Mac OS 8/9), sound (for Mac OS 8/9) and string handles previously obtained. The next line specifies that no response function is required to be executed when the notification is posted.

doIdle

`doIdle` is called when the installed timer fires.

If the user has not just chosen the Notification item in the Demonstration menu (`gNotificationDemoInvoked` is false), `doIdle` simply returns immediately.

If, however, that item has just been chosen, and if 10 seconds (600 ticks) have elapsed since that choice was made, the following occurs:

- The calls to `GetFrontProcess` and `SameProcess` determine whether the current foreground process is Miscellany. If it is not, the notification request is installed in the notification queue by `NMInstall` and the global variable `gNotificationInQueue` is set to indicate that a request has been placed in the queue by Miscellany. (This latter causes `doDisplayMessageToUse`r to be called when the `kEventAppActivated` event is received. `doDisplayMessageToUser` removes the notification request from the queue and has Miscellany convey the required message to the user.) Also, `gNotificationDemoInvoked` is set to false so as to ensure that the main if block only executes once after the Notification item is chosen.
- If, however, the current foreground process is Miscellany, the function `doDisplayMessageToUser` is called to present the required message to the user in the normal way. Once again `gNotificationDemoInvoked` is reset to false so as to ensure that the main if block only executes once after the Notification item is chosen.

doDisplayMessageToUser

`doDisplayMessageToUser` is called by `appEventHandler` and doIdle in the circumstances previously described.

If a Miscellany notification request is in the queue, `NMRemove` removes it from the queue and `gNotificationInQueue` is set to false to reflect this condition. (Recall that, if the nmResp field of the notification structure is not assigned -1, the application itself must remove the queue element from the queue.)

Regardless of whether there was a notification in the queue or not, Miscellany then presents its alert. When the alert is dismissed, the notification's icon suite, sound and string resources are released/disposed of.

ProgressBar.c

doProgressBar

`doProgressBar` is called when the user chooses Determinate Progress Indicator from the Demonstration menu.

`GetNewDialog` creates a modal dialog. The call to `GetDialogItemAsControl` retrieves the dialog's progress indicator control. `SetControlMaximum` sets the control's maximum value to equate to the number of steps in a simulated time-consuming task.

The main for loop performs the simulated time-consuming task, represented to the user by the drawing of a large number of colored rectangles in the window. The task involves 3456 calls to FrameRect.

Within the inner for loop, `CheckEventQueueForCancel` is called to check whether the user has pressed the Command-period key. If so, a `'snd '` resource is loaded, played, and released, the dialog is disposed of, an advisory message in drawn in the window, and the function returns.

Each time around the inner for loop, a progress indicator control's value is incremented.

When the outer for loop exits (that is, when the Command-period key combination is not pressed before the simulated time-consuming task completes), the dialog is disposed of.

ColorPicker.c

(f)`doColorPicker` is called when the user chooses Color Picker from the Demonstration menu.

The first block erases the window's content area and paints a rectangle in the color that will be passed in GetColor's inColor parameter.

The next line assigns 0 to the fields of the Point variable to be passed in GetColor's where parameter. ((0,0) will cause the Color Picker dialog to be centerd on the main screen.)

The call to `GetColor` displays the Color Picker's dialog. `GetColor` retains control until the user clicks either the OK button or the Cancel button, at which time the port rectangle is invalidated, causing the function `doDrawColorPickerChoice` to be called.

doDrawColorPickerChoice

If the user clicked the OK button, a filled rectangle is painted in the window in the color returned in GetColor's outColor parameter, and the values representing the red, green, and blue components of this color are displayed at the top of the window in hexadecimal. Note that the function `doDecimalToHexadecimal` is called to convert the decimal (UInt32) values in the fields of the RGBColor variable outColor to hexadecimal.

If the user clicks the Cancel button, a filled rectangle is painted in the window in the color passed in `GetColor`'s `inColor` parameter.

doDecimalToHexadecimal

`doDecimalToHexadecimal` converts a UInt16 value to a hexadecimal string.

MultiMonitor.c

deviceLoopDraw

`deviceLoopDraw` is the image-optimizing drawing function the universal procedure pointer to which is passed in the second parameter in the `DeviceLoop` call. (Recall that the `DeviceLoop` call is made whenever the Multiple Monitors Draw item in the Demonstration menu has been selected and an `kEventDrawContent` event type is received.) `DeviceLoop` scans all active video devices, calling `deviceLoopDraw` whenever it encounters a device that intersects the drawing region, and passing certain information to `deviceLoopDraw`.

The second line casts the `SInt32` value received in the `userData` parameter to a `WindowRef`. The window's content area is then erased.

If an examination of the device's attributes, as received in the `deviceFlags` formal parameter, reveals that the device is a color device, three rectangles are painted in the window in three different colors. (The luminance value of these colors is the same, meaning that the rectangles would all be the same shade of gray if they were drawn on a monochrome (grayscale) device.)

If the examination of the device's attributes reveals that the device is a monochrome device, the rectangles are painted in three distinct shades of gray.

HelpTag.c

doHelpTagControl and doHelpTagWindow

`doHelpTagControl` and `doHelpTagWindow` create Help tags for the bevel button control and the window.

The call to memset clears the specified block of memory. The call to `HMSetTagDelay` sets the delay, in milliseconds, before the tag opens.

For the bevel button, the `tagSide` field of the `HMHelpContentRec` structure is assigned a value which will cause the control's tag to be displayed below the control with its left side aligned with the left side of the button. For the window, the `tagSide` field is assigned a value that will cause the control's tag to be displayed on the window's right, centered vertically.

The main block sets the content type and retrieves and assigns the minimum and maximum content strings from a `'STR#'` resource. The calls to `HMSetControlHelpConten`t and `HMSetWindowHelpContent` install the Help tags on the control and window.

Chapter 26. More on Text-Multilingual Text Engine

IN THIS CHAPTER

- Introduction
- Working with MLTE
- Main Constants, Data Types, and Functions
- Demonstration Program MLTETextEditor Listing
- Demonstration Program MLTETextEditor Comments

Introduction

The Multilingual Text Engine (MLTE), which was introduced with Mac OS 9.0 as an alternative for TextEdit, profoundly simplified the task of handling multistyled text and provided many new features not provided by TextEdit. The main additional features are as follows:

- Documents can be larger than 32KB (the TextEdit limit).
- Tabs.
- Text justification.
- Ability to embed movie, sound, and graphics objects in documents.
- Built-in support for
    - Scroll bar creation and handling, including live scrolling and proportional scroll boxes.
    - Undo/redo (32 levels).
    - Printing.
    - Drag and drop.
    - The public clipboard, specifically, copying and pasting:

        Plain text.

        Plain text with style resources.

        Unicode text.

        Flattened Unicode style information.

        Movies, graphics, and sound.

        In-line input.

MLTE uses Apple Type Services for Unicode Imaging (**ATSUI**) to measure and draw text. ATSUI, which was introduced with Mac OS 8.6, replaced QuickDraw and the Script Manager as the low-level means of imaging and measuring text.

MLTE renders text into a rectangular frame. Applications can specify that lines be arbitrarily wide or auto-wrapped.

Global Layout Settings

The following are the main settings applying to the whole document:

- Justification, which can be default, left, right, center, full, or forced full.
- Tab values.
- Margins.
- Auto-indent on or off.
- Text auto-wrap on or off.
- Read-only status on or off.
- Line direction.

Undoable and Redoable Actions

The following actions are undoable and redoable:

- Typing.
- Cut, paste, and clear.
- Change font, and font size, style, and color.
- Justification.
- Drag-and-drop move and copy.

Selection Behavior

Within an MLTE document, a single-click defines an insertion point. (Recall from Chapter 21 that an insertion point is, in effect, a selection containing zero characters.) A double-click selects a word, and a triple-click selects a line.

File Types

MLTE supports saving and opening files of the following types:

- A new type introduced with MLTE called Textension (`'txtn'`). This should be the preferred type for media-rich documents that can contain movies, graphics, and sound.
- Text (`'TEXT'`), with or without style information. Style information may be saved as either `'styl'` resources or `'MPSR'` resources. If `'styl'` resources are used, documents can have text in an unlimited number of styles; however, tabs will not be saved. If `'MPSR'` resources are used , only the first style in the document will be saved.

Note

For example, SimpleText saves style information in *`'styl'`* resources whereas Macintosh Programmer's Workshop (MPW), CodeWarrior and

BBEdit save style information in MPW `'MPSR'` resources.

- Plain Unicode text (`'utxt'`).
- Movie (`'MooV'`), sound (`'sfil'` and `'AIFF'`), and picture (`'PICT'`).

Working with MLTE

Initializing and Terminating MLTE

`TXNInitTextension` should be called at program start to initialise the Textension library:

```
OSStatus  TXNInitTextension(const TXNMacOSPreferredFontDescription iDefaultFonts[],
                            ItemCount iDefaultFonts,TXNInitOptions iUsageFlags);
```

`iDefaultFonts` `NULL`, or a table of entries for any encoding for which it is desired to designate a default font:

```
struct TXNMacOSPreferredFontDescription
{
  UInt32       fontID;     // Can assign kTXNDefaultFontName
  Fixed        pointSize;  // Can assign kTXNDefaultFontSize
  TextEncoding encoding;
  Style        fontStyle;  // Can assign kTXNDefaultFontStyle
} ;
```

For the `encoding` field, relevant constants are

```
kTXNSystemDefaultEncoding

kTXNMacOSEncoding

kTXNUnicodeEncoding
```

`kTXNSystemDefaultEncoding` the encoding preferred by MLTE and the system. This is Unicode if ATSUI is present, as it will be on Mac OS 8.6 and later.)

`iDefaultFonts` Number of default fonts designated.

`IUsageFlags` Specifies whether movies, sound, and/or graphics should be supported. Relevant constants are:

```
kTXNWantMoviesMask

kTXNWantSoundMask

kTXNWantGraphicsMask
```

At program termination, you should call `TXNTerminateTextension` to close the Textension library and perform other clean-up actions.

Allocating and Deleting a TXNObject

Ordinarily, to create a new document, you create a new window and then pass a reference to that window in a call to `TXNNewObject`. `TXNNewObject` creates a new `TXNObject`, an object that contains private variables and functions required to handle text formatting:

```
OSStatus  TXNNewObject(const FSSpec *               iFileSpec,       // Can be NULL
                       WindowRef                    iWindow,
                       Rect *                       iFrame,          // Can be NULL
                       TXNFrameOptions              iFrameOptions,
```

```
                                TXNFrameType                iFrameType,
                                TXNFileType                 iFileType,
                                TXNPermanentTextEncodingType iPermanentEncoding,
                                TXNObject *                 oTXNObject,
                                TXNFrameID *                oTXNFrameID,
                                TXNObjectRefcon             iRefCon);
```

`IFileSpec` Pointer to file system specification structure. The file is read in, and its contents displayed, after the `TXNObject` is allocated. If `NULL` is passed in this parameter, the document will be empty at start.

`IWindow` Reference to the window to be attached to the `TXNObject`, and in which the document will be displayed.

`IFrame` The area of the window in which the document's contents are to be displayed. Passing `NULL` in this parameter sets the frame to equate to the window's port rectangle.

`IFrameOptions` The options supported by this frame. The principal relevant constants are

`kTXNShowWindowMask` Show window before `TXNNewObject` returns.

`kTXNWantHScrollBarMask` Include horizontal scroll bar.

`kTXNWantVScrollBarMask` Include vertical scroll bar.

`kTXNNoSelectionMask` Do not display insertion point.

`IFrameType` The frame type. Relevant constants are

`kTXNTextEditStyleFrameType`

`kTXNPageFrameType`

`kTXNMultipleFrameType`

`iFileType` The primary file type. The principal relevant constants are:

`kTXNTextensionFile`

`kTXNTextFile`

`kTXNUnicodeTextFile`

If you specify `kTXNTextFile`, and want style information to be saved, you can specify whether that information should be saved in a `'styl'` resource or an `'MPSR'` resource when calling the function `TXNSave` (see below).

`iPermanentEncoding` The encoding the application considers text to be in. Relevant constants are

`kTXNSystemDefaultEncoding` Encoding preferred by MLTE and the system. (This is Unicode if ATSUI is present.)

`KTXNMacOSEncoding` Incoming and outgoing text to be in traditional MacOS encoding.

`kTXNUnicodeEncoding` Incoming and outgoing text to be in Unicode, even on systems that do not have ATSUI.

`OTXNObject` On output, a pointer to a `TXNObject`.

`OTXNFrameID` On output, a unique ID for the frame.

`IRefCon` Reference constant for use by the application.

If `TXNNewObject` is called with `NULL` passed in the `iWindow` parameter, a window can later be attached to the `TXNObject` by a call to the function `TXNAttachObjectToWindow`.

A previously allocated `TXNObject` and its associated data structures may be deleted by a call to the function `TXNDeleteObject`.

Setting and Getting Global Layout Settings

As previously stated, certain layout settings (for example, justification, tabs, and margins) apply to the whole `TXNObject`, that is, the whole document. These layout settings are referred to as **control information.** You can set control information by calling `TXNSetTXNObjectControls`:

```
OSStatus  TXNSetTXNObjectControls(TXNObject iTXNObject,
                                  Boolean iClearAll,
                                  ItemCount iControlCount,
                                  TXNControlTag iControlTags[],
                                  TXNControlData iControlData[]);
```

`IClearAll` Pass `true` to reset all controls to the default.

`IControlCount` The number of elements in the `iControlTags` and `iControlData` arrays, that is, the number of settings being changed.

`IControlTags` An array of type `TXNControlTag` containing control tags. The principal relevant tags are:

```
kTXNJustificationTag

kTXNTabSettingsTag

kTXNMarginsTag

kTXNWordWrapStateTag
```

`iControlData` An array of `TXNControlData` structures containing the control information being set. The `TXNControlData` structure, and its associated structures, are as follows:

```
union TXNControlData
{
  UInt32       uValue;
  SInt32       sValue;
  TXNTab       tabValue;
  TXNMargins * marginsPtr;
} ;
typedef union TXNControlData;

struct TXNTab
{
  SInt16     value;
  TXNTabType tabType;
  UInt8      filler;
} ;
typedef struct TXNTab TXNTab;
```

```
struct TXNMargins
{
  SInt16 topMargin;
  SInt16 leftMargin;
  SInt16 bottomMargin;
   SInt16 rightMargin;
} ;
typedef struct TXNMargins TXNMargins;
```

Constants relevant to the `uValue` field when setting justification are

`kTXNFlushDefault`

`kTXNFlushLeft`

`kTXNFlushRight`

`kTXNCenter`

`kTXNFullJust`

`kTXNForceFullJust`

Constants relevant to the `tabType` field when setting tabs are

`kTXNRightTab`

`kTXNLeftTab`

`kTXNCenterTab`

Constants relevant to the `uValue` field when setting word wrapping are

`kTXNAutoWrap`

`kTXNNoAutoWrap`

You can get control information by calling `TXNGetTXNObjectControls`:

```
OSStatus  TXNGetTXNObjectControls(TXNObject      iTXNObject,
                                  ItemCount      iControlCount,
                                  TXNControlTag  iControlTags[],
                                  TXNControlData oControlData[]);
```

`IControlCount` The number of elements in the `iControlTags` and `iControlData` arrays.

`IControlTags` An array of type `TXNControlTag` containing control tags.

`OControlData` An array of `TXNControlData` structures containing, on output, the control information requested by the tags in the `iControlTags` array.

Setting the Background

You can set the background by calling `TXNSetBackground`:

```
OSStatus  TXNSetBackground(TXNObject iTXNObject,TXNBackground * iBackgroundInfo);
```

`IBackgroundInfo` A pointer to a `TXNBackground` structure that describes the background. The `TXNBackground` structure and its associated union are as follows:

```
struct TXNBackground
{
  TXNBackgroundType bgType;  // Assign kTXNBackgroundTypeRGB
  TXNBackgroundData bg;
} ;
```

```
typedef struct TXNBackground TXNBackground;

union TXNBackgroundData
{
  RGBColor color;
} ;
typedef union TXNBackgroundData TXNBackgroundData;
```

The only background type available with Version 1.1 of MLTE is a color. `TXNBackgroundData` is a union so that it can be expanded in the future to support other background types, such as pictures.

Setting and Getting Type Attributes

You can set type attributes such as text size, style, and color by calling `TXNSetTypeAttributes`:

```
OSStatus  TXNSetTypeAttributes(TXNObject         iTXNObject,
                               ItemCount         iAttrCount,
                               TXNTypeAttributes iAttributes[],
                               TXNOffset         iStartOffset,
                               TXNOffset         iEndOffset);
```

`IAttrCount` The number of elements in the `iAttributes` array, that is, the number of attributes being set.

`IAttributes` An array of `TXNTypeAttributes` structures specifying the attribute being set and the data, or pointer to the data, that will set the attribute. Values less than or equal to `sizeof(UInt32)` are passed by value. Values greater than `sizeof(UInt32)` are passed by pointer. The `TXNTypeAttributes` structure, and its main associated structure, are as follows:

```
struct TXNTypeAttributes
{
  TXTNTag          tag;
  ByteCount        size;
  TXNAttributeData data;
} ;
typedef struct TXNTypeAttributes TXNTypeAttributes;

union TXNAttributeData
{
  void *              dataPtr;
  UInt32              dataValue;
  TXNATSUIFeatures *  atsuFeatures;
  TXNATSUIVariations * atsuVariations;
} ;
typedef union TXNAttributeData TXNAttributeData;
```

The principal constants relevant to the `tag` field are

```
kTXNQDFontSizeAttribute

kTXNQDFontStyleAttribute

kTXNQDFontColorAttribute
```

The associated constants relevant to the `size` field are

```
kTXNFontSizeAttributeSize

kTXNQDFontStyleAttributeSize
```

kTXNQDFontColorAttributeSize

iStartOffset The offset at which to begin setting the attributes. If the requirement is to apply the attributes to the current selection, pass kTXNUseCurrentSelection in this parameter.

IEndOffset The offset at which to end setting the attributes. This parameter is ignored if kTXNUseCurrentSelection is passed in the iStartOffset parameter.

When your application detects a mouse-down in the menu bar or a Command-key combination, it typically adjusts its menus, enabling and disabling menu items as appropriate. If your application contains menus that allow the user to set type attributes, it must also prepare those menus for display by checkmarking and un-checkmarking items as appropriate.

Using a Size menu as an example, if the current selection (whether it be an empty or non-empty selection) contains text that is all of a single size, the menu item corresponding to that size, and only that menu item, should be checkmarked before the menu is displayed. However, if the selection contains text in two or more sizes, all menu items should be uncheckmarked. It is thus necessary to examine the selection to determine whether it contains a continuous size run or multiple sizes.

You can examine the current selection to determine whether font size, style, and color are continuous by calling TXNGetContinuousTypeAttributes:

```
OSStatus  TXNGetContinuousTypeAttributes(TXNObject            iTxnObject,
                                         TXNContinuousFlags * oContinuousFlags,
                                         ItemCount            iCount,
                                         TXNTypeAttributes    ioTypeAttributes[]);
```

OContinuousFlags On output, the relevant bit, or bits, of this parameter can be examined to determine whether the associated attribute, or attributes, is/are continuous. For example, if size is continuous, bit 1 will be set. The relevant constants are

kTXNSizeContinuousMask

kTXNStyleContinuousMask

kTXNColorContinuousMask

iCount Number of elements in the ioTypeAttributes array, that is, the number of attributes being examined.

IoTypeAttributes An array of TXNTypeAttributes structures (see above). On input, the tag field in each structure specifies the attribute to be examined. On output, if the attribute is continuous, the dataValue or dataPtr field of the data field will contain a value, or a pointer to data, which can be used to determine which menu item should be checkmarked.

Functions Relevant to Events

The following functions are relevant to event handling:

Function	*Description*
TXNClick	Processes clicks in the content region, handling text selection, scrolling, drag and drop, and playing movies and sound.
TXNUpdate	Handles update events, redrawing the contents of the window. Calls BeginUpdate and EndUpdate.
TXNForceUpdate	Forces an update event to be generated, thus forcing the contents of the window to be redrawn.
TXNDraw	Similar to TXNUpdate, except that BeginUpdate and EndUpdate are not called. Should be used, in lieu of TXNUpdate, for a window that contains multiple TXNObjects or some graphic element.

`TXNActivate`	In MLTE, activation is a two-step process. `TXNActivate` performs the first step, which has to do with activating or deactivating the scroll bars. When `true` is passed in the `TXNScrollBarState` parameter, the scroll bars are activated, even when text input (selection and typing) has been defeated by `TXNFocus` (see below). When `false` is passed in the `TXNScrollBarState` parameter, the scroll bars are deactivated.
`TXNFocus`	The second step in the activation process has to do with activating text input (selection and typing). When `true` is passed in the `iBecomingFocused` parameter, text input is activated. When `false` is passed in the `iBecomingFocused` parameter text input is deactivated.
`TXNAdjustCursor`	Handles cursor shape-changing. If the cursor is over a text area, it is set to the I-beam shape. If it is over a scrollbar, a movie, graphic, or a sound, or outside the frame, it is set to the arrow shape. Not relevant in applications using the Carbon event model.
`TXNZoomWindow`	Handles mousedowns in the zoom box.
`TXNGrowWindow`	Handles mouse-downs in the size box.
`TXNGetSleepTicks`	Gets the appropriate sleep time.

In Carbon applications, calling `TXNKeyDown` on receipt of key events is not necessary, since MLTE handles the event itself without any assistance on the part of your application. Note also that

- When the Classic event model is used, it is not possible to filter key events because the event is sent to MLTE before `WaitNextEvent` delivers it to your application.
- When the Carbon event model is used, key events are sent through the Carbon event system before they are sent to MLTE. This means that you can filter key events by installing a handler for the `kEventUnicodeForKeyEvent` event kind, for example (`kEventClassTextInput` event class).

Functions Relevant to the File Menu

The following functions are relevant to your application's File menu:

Function	***Description***
`TXNSave`	Saves the contents of a document to a file of a specified file type (for example, Textension, text, or Unicode text). The data fork of the file must be open and its file reference number passed in the `iDataReference` parameter. For files of type text, if style information is also to be saved, the resource fork of the file must also be open and its file reference number passed in the `iResourceReference` parameter (The `iResourceReference` parameter is ignored when the Textension file type is specified.) `kTXNMultipleStylesPerTextDocumentResType` passed in the `iResType` parameter causes style information be saved to a `'style'` resource. To specify that style information be saved to an `'MPSR'` resource, pass `kTXNSingleStylePerTextDocumentResType` in the `iResType` parameter. `TXNSave` does not move the file mark before writing to a file. This allows you to write private data first (if required), followed by the data written by `TXNSave`.
`TXNRevert`	Reverts to the last saved version of the document or, if the document has never been saved, reverts to an empty document.

`TXNPageSetup`	Displays the Page Setup dialog and reformats the text if settings are changed by the user.
`TXNPrint`	Displays the Print dialog and prints the document.
`TXNGetChangeCount`	Returns the number of times the document has been changed since the last save or, for new documents which have not yet been saved, since the document was created. Useful for determining whether the Save and Revert items should be enabled or disabled.
`TXNDataSize`	Returns the size in bytes of the characters in the document. Useful for determining if the Save As, Page Setup and Print items should be enabled or disabled.

Functions Relevant to the Edit Menu

The following functions are relevant to your application's Edit menu:

Function	*Description*
`TXNCanUndo`	Returns `true` if the last action is undoable, in which case the Undo item should be enabled.
`TXNCanRedo`	Returns `true` if the last action is redoable, in which case the Redo item should be enabled.
`TXNUndo`	Undoes the last action.
`TXNRedo`	Re-does the last action.
`TXNCut`	Cuts the current selection to the MLTE private scrap.
`TXNCopy`	Copies the current selection to the MLTE private scrap.
`TXNPaste`	Pastes the MLTE private scrap to the document.
`TXNClear`	Clears the current selection.
`TXNSelectAll`	Selects everything in a frame.
`TXNIsSelectionEmpty`	Returns `false` if the current selection is not empty, in which case the Cut, Copy, and Clear items should be enabled.
`TXNIsScrapPastable`	Returns `true` if the current MLTE scrap is pastable, in which case the Paste item should be enabled.
`TXNConvertToPublicScrap`	Converts the MLTE private scrap to the public clipboard. In Carbon applications, this function should not be called on suspend events. Typically, it should be called immediately after `TXNCut` and `TXNCopy`.
`TXNDataSize`	Returns the size in bytes of the characters in the document. Useful for determining if the Select All item should be enabled or disabled.

Note that, in Carbon applications, there is no need to call `TXNConvertFromPublicScrap` to convert the public clipboard to the MLTE private scrap. In Carbon applications, MLTE automatically keeps the public scrap and private scrap synchronised.

More on TXNCanUndo and TXNCanRedo

When `TXNCanUndo` and `TXNCanRedo` return `true`, they also return in their `oTXNActionKey` parameters an **action key,** which can be used by your application to load an indexed string describing the undoable or re-doable action. The Undo and Redo items should be set to this string using `SetMenuItemText`. The following are the main action key constants:

`kTXNTypingAction`	`kTXNChangeFontColorAction`	`kTXNAlignLeftAction`
`kTXNCutAction`	`kTXNChangeFontSizeAction`	`kTXNAlignRightAction`
`kTXNPasteAction`	`kTXNChangeStyleAction`	`kTXNAlignCenterAction`

```
kTXNClearAction          kTXNMoveAction                kTXNUndoLastAction
kTXNChangeFontAction     kTXNDropAction
```

Creating, Preparing, and Handling the Font Menu

MLTE contains functions that greatly simplify the task of creating and managing the Font menu.

The `TXNNewFontMenuObject` function may be used to create a hierarchical Font menu. A reference to the (empty) Font menu, the menu ID, and the starting ID for the sub-menus are passed in the first three parameters, and a pointer to a `TXNFontMenuObject` is returned in the fourth parameter. The starting ID for the sub-menus must be 160 or higher.

To prepare the Font menu for display when the user clicks in the menu bar, you should simply call `TXNPrepareFontMenu`, which checks and unchecks items in the Font menu as appropriate.

A call to `TXNDoFontMenuSelection`, with the `TXNFontMenuObject` obtained by the call to `TXNNewFontMenuObject` passed in the second parameter and the menu ID and chosen menu item passed in the third and fourth parameters, sets the chosen font and changes the current selection to that font.

At program termination, you should call `TXNDisposeFontMenuObject` to dispose of the `TXNFontMenuObject` and its menu handle.

Setting Data

You can replace a specified range with data by calling `TXNSetData`.

```
OSStatus  TXNSetData(TXNObject iTXNObject,TXNDataType iDataType,void * iDataPtr,
                     ByteCount iDataSize,TXNOffset iStartOffset,TXNOffset iEndOffset);
```

`IDataType` The type of data. Relevant constants are

```
kTXNTextData
kTXNPictureData
kTXNMovieData
kTXNSoundData
kTXNUnicodeTextData
```

`iDataPtr` A pointer to the new data.

`IDataSize` The size of new data.

`IStartOffset` offset to the beginning of the range to replace.

`IEndOffset` The offset to the end of range to replace.

You can replace a specified range with the contents of a specified file by calling `TXNSetDataFromFile`.

```
OSStatus  TXNSetDataFromFile(TXNObject iTXNObject,SInt16 iFileRefNum,OSType iFileType,
                             ByteCount iFileLength,TXNOffset iStartOffset,
                             TXNOffset iEndOffset);
```

`IFileRefNum` The file reference number.

`IFileType` The file type. For file types supported by MLTE, the relevant constants are

```
kTXNTextensionFile
kTXNTextFile
kTXNPictureFile
```

`kTXNMovieFile`
`kTXNSoundFile`
`kTXNAIFFFile`
`kTXNUnicodeTextFile`

`iFileLength` How much data should be read. Ignored if the file type is a file type that MLTE supports.

This parameter is useful when you want data that is embedded in the file. For the whole file, pass `kTXNEndOffset` in this parameter.

`IStartOffset` The offset to the beginning of the range to replace.

`IEndOffset` The offset to the end of range to replace.

The data fork of the file must be opened and the file mark set by the application. MLTE does not move the file's marker before reading the data.

Main Constants, Data Types, and Functions

Constants

Initializing

```
kTXNWantMoviesMask           = 1L << kTXNWantMoviesBit
kTXNWantSoundMask            = 1L << kTXNWantSoundBit
kTXNWantGraphicsMask         = 1L << kTXNWantGraphicsBit
```

Text Encoding

```
kTXNSystemDefaultEncoding    = 0
kTXNMacOSEncoding            = 1
kTXNUnicodeEncoding          = 2
```

Frame Options

```
kTXNShowWindowMask           = 1L << kTXNShowWindowBit
kTXNWantHScrollBarMask       = 1L << kTXNWantHScrollBarBit
kTXNWantVScrollBarMask       = 1L << kTXNWantVScrollBarBit
kTXNNoSelectionMask          = 1L << kTXNNoSelectionBit
```

Frame Types

```
kTXNTextEditStyleFrameType   = 1
kTXNPageFrameType            = 2
kTXNMultipleFrameType        = 3
```

Global Layout

```
kTXNJustificationTag         = FOUR_CHAR_CODE('just')
kTXNTabSettingsTag           = FOUR_CHAR_CODE('tabs')
kTXNMarginsTag               = FOUR_CHAR_CODE('marg')
kTXNWordWrapStateTag         = FOUR_CHAR_CODE('wwrs')
kTXNFlushDefault             = 0
kTXNFlushLeft                = 1
kTXNFlushRight               = 2
kTXNCenter                   = 4
kTXNFullJust                 = 8
kTXNForceFullJust            = 16
kTXNAutoWrap                 = false
kTXNNoAutoWrap               = true
kTXNRightTab                 = -1
kTXNLeftTab                  = 0
kTXNCenterTab                = 1
```

Setting the Background

```
kTXNBackgroundTypeRGB        = 1
```

Type Attributes

```
kTXNSizeContinuousMask       = 1L << kTXNSizeContinuousBit
kTXNStyleContinuousMask      = 1L << kTXNStyleContinuousBit
kTXNColorContinuousMask      = 1L << kTXNColorContinuousBit
kTXNQDFontSizeAttribute      = FOUR_CHAR_CODE('size')
kTXNQDFontStyleAttribute     = FOUR_CHAR_CODE('face')
kTXNQDFontColorAttribute     = FOUR_CHAR_CODE('klor')
```

```
kTXNFontSizeAttributeSize    = sizeof(Fixed)
kTXNQDFontStyleAttributeSize = sizeof(Style)
kTXNQDFontColorAttributeSize = sizeof(RGBColor)
```

Data Types

```
kTXNTextData                 = FOUR_CHAR_CODE('TEXT')
kTXNPictureData              = FOUR_CHAR_CODE('PICT')
kTXNMovieData                = FOUR_CHAR_CODE('moov')
kTXNSoundData                = FOUR_CHAR_CODE('snd ')
kTXNUnicodeTextData          = FOUR_CHAR_CODE('utxt')
```

File Types

```
kTXNTextensionFile           = FOUR_CHAR_CODE('txtn')
kTXNTextFile                 = FOUR_CHAR_CODE('TEXT')
kTXNPictureFile              = FOUR_CHAR_CODE('PICT')
kTXNMovieFile                = FOUR_CHAR_CODE('MooV')
kTXNSoundFile                = FOUR_CHAR_CODE('sfil')
kTXNAIFFFile                 = FOUR_CHAR_CODE('AIFF')
kTXNUnicodeTextFile          = FOUR_CHAR_CODE('utxt')
```

Undo/Redo Action Keys

```
kTXNTypingAction             = 0
kTXNCutAction                = 1
kTXNPasteAction              = 2
kTXNClearAction              = 3
kTXNChangeFontAction         = 4
kTXNChangeFontColorAction    = 5
kTXNChangeFontSizeAction     = 6
kTXNChangeStyleAction        = 7
kTXNAlignLeftAction          = 8
kTXNAlignCenterAction        = 9
kTXNAlignRightAction         = 10
kTXNDropAction               = 11
kTXNMoveAction               = 12
kTXNUndoLastAction           = 1024
```

Saving Text Files-Style Information

```
kTXNSingleStylePerTextDocumentResType    = FOUR_CHAR_CODE('MPSR')
kTXNMultipleStylesPerTextDocumentResType = FOUR_CHAR_CODE('styl')
```

Data Types

```
typedef struct OpaqueTXNObject*         TXNObject;
typedef struct OpaqueTXNFontMenuObject* TXNFontMenuObject;
typedef UInt32                          TXNFrameID;
typedef OptionBits                      TXNInitOptions;
typedef UInt32                          TXNPermanentTextEncodingType;
typedef OptionBits                      TXNFrameOptions;
typedef UInt32                          TXNFrameType;
typedef UInt32                          TXNBackgroundType;
typedef FourCharCode                    TXNControlTag;
typedef SInt8                           TXNTabType;
typedef OptionBits                      TXNContinuousFlags;
typedef ByteCount                       TXNTypeRunAttributeSizes;
typedef OSType                          TXNDataType;
typedef OSType                          TXNFileType;
typedef UInt32                          TXNActionKey;
```

Global Layout

```
union TXNControlData
{
  UInt32       uValue;
  SInt32       sValue;
  TXNTab       tabValue;
  TXNMargins * marginsPtr;
} ;
typedef union TXNControlData TXNControlData;

struct TXNTab
{
  SInt16     value;
  TXNTabType tabType;
  UInt8      filler;
} ;
typedef struct TXNTab TXNTab;

struct TXNMargins
{
  SInt16 topMargin;
  SInt16 leftMargin;
  SInt16 bottomMargin;
  SInt16 rightMargin;
} ;
typedef struct TXNMargins TXNMargins;
```

Setting the Background

```
struct TXNBackground
{
  TXNBackgroundType bgType;
  TXNBackgroundData bg;
} ;
typedef struct TXNBackground TXNBackground;

union TXNBackgroundData
{
  RGBColor color;
} ;
typedef union TXNBackgroundData TXNBackgroundData;
```

Type Attributes

```
struct TXNTypeAttributes
{
    TXTNTag          tag;
    ByteCount        size;
    TXNAttributeData data;
} ;
typedef struct TXNTypeAttributes TXNTypeAttributes;

union TXNAttributeData
{
  void *               dataPtr;
  UInt32               dataValue;
  TXNATSUIFeatures *   atsuFeatures;
  TXNATSUIVariations * atsuVariations;
} ;
typedef union TXNAttributeData TXNAttributeData;
```

Font Description

```
struct TXNMacOSPreferredFontDescription
{
```

```
  UInt32       fontID;
  Fixed        pointSize;
  TextEncoding encoding;
  Style        fontStyle;
} ;
typedef struct TXNMacOSPreferredFontDescription TXNMacOSPreferredFontDescription;
```

Functions

Initializing and Terminating

```
void      TXNTerminateTextension(void);
OSStatus  TXNInitTextension(const TXNMacOSPreferredFontDescription iDefaultFonts[],
          ItemCount iCountDefaultFonts,TXNInitOptions iUsageFlags);
```

Allocating and Deleting TXNObject

```
OSStatus  TXNNewObject(const FSSpec *iFileSpec,WindowRef iWindow,Rect *iFrame,
          TXNFrameOptions iFrameOptions,TXNFrameType iFrameType, TXNFileType iFileType,
          TXNPermanentTextEncodingType iPermanentEncoding,TXNObject *oTXNObject,
          TXNFrameID *oTXNFrameID,TXNObjectRefcon iRefCon);
void      TXNDeleteObject(TXNObject iTXNObject);
```

Attaching an Object to a Window

```
OSStatus  TXNAttachObjectToWindow(TXNObject iTXNObject,GWorldPtr iWindow,
          Boolean iIsActualWindow);
Boolean   TXNIsObjectAttachedToWindow(TXNObject iTXNObject);
```

Resizing the Frame

```
void      TXNResizeFrame(TXNObject iTXNObject,UInt32 iWidth,UInt32 iHeight,
          TXNFrameID iTXNFrameID);
```

Setting and Getting Global Layout Settings

```
OSStatus  TXNSetTXNObjectControls(TXNObject iTXNObject,Boolean iClearAll,
          ItemCount iControlCount,TXNControlTag iControlTags[],TXNControlData iControlData
[]);
OSStatus  TXNGetTXNObjectControls(TXNObject iTXNObject,ItemCount iControlCount,
          TXNControlTag iControlTags[],TXNControlData oControlData[]);
```

Setting the Background

```
OSStatus  TXNSetBackground(TXNObject iTXNObject,TXNBackground *iBackgroundInfo);
```

Setting and Getting Type Attributes

```
OSStatus  TXNSetTypeAttributes(TXNObject iTXNObject,ItemCount iAttrCount,
          TXNTypeAttributes iAttributes[],TXNOffset iStartOffset,TXNOffset iEndOffset);
OSStatus  TXNGetContinuousTypeAttributes(TXNObject iTxnObject,
          TXNContinuousFlags *oContinuousFlags,ItemCount iCount,
          TXNTypeAttributes  ioTypeAttributes[]);
```

Event Handling

```
void      TXNClick(TXNObject iTXNObject,const EventRecord *iEvent);
void      TXNUpdate(TXNObject iTXNObject);
void      TXNForceUpdate(TXNObject iTXNObject);
void      TXNDraw(TXNObject iTXNObject,GWorldPtr iDrawPort);
OSStatus  TXNActivate(TXNObject iTXNObject,TXNFrameID iTXNFrameID,
          TXNScrollBarState iActiveState);
```

```
void      TXNFocus(TXNObject iTXNObject,Boolean iBecomingFocused);
void      TXNAdjustCursor(TXNObject iTXNObject,RgnHandle ioCursorRgn);
void      TXNZoomWindow(TXNObject iTXNObject,short iPart);
void      TXNGrowWindow(TXNObject iTXNObject,const EventRecord *iEvent);
UInt32    TXNGetSleepTicks(TXNObject iTXNObject);
```

Functions Relevant to the File Menu

```
OSStatus  TXNSave(TXNObject iTXNObject,TXNFileType iType,OSType iResType,
          TXNPermanentTextEncodingType iPermanentEncoding,FSSpec *iFileSpecification,
          SInt16 iDataReference,SInt16 iResourceReference);
OSStatus  TXNRevert(TXNObject iTXNObject);
OSStatus  TXNPageSetup(TXNObject iTXNObject);
OSStatus  TXNPrint(TXNObject iTXNObject);
```

Functions Relevant to the Edit Menu

```
Boolean   TXNCanUndo(TXNObject iTXNObject,TXNActionKey *oTXNActionKey);
Boolean   TXNCanRedo(TXNObject iTXNObject,TXNActionKey *oTXNActionKey);
void      TXNUndo(TXNObject iTXNObject);
void      TXNRedo(TXNObject iTXNObject);
OSStatus  TXNCut(TXNObject iTXNObject);
OSStatus  TXNCopy(TXNObject iTXNObject);
OSStatus  TXNPaste(TXNObject iTXNObject);
OSStatus  TXNClear(TXNObject iTXNObject);
void      TXNSelectAll(TXNObject iTXNObject);
```

Creating, Preparing, Handling, and Disposing of the Font Menu

```
OSStatus  TXNNewFontMenuObject(MenuRef iFontMenuHandle,SInt16 iMenuID,
          SInt16 iStartHierMenuID,TXNFontMenuObject *oTXNFontMenuObject);
OSStatus  TXNPrepareFontMenu(TXNObject iTXNObject,TXNFontMenuObject iTXNFontMenuObject);
OSStatus  TXNDoFontMenuSelection(TXNObject iTXNObject,TXNFontMenuObject iTXNFontMenuObject,
          SInt16 iMenuID,SInt16 iMenuItem);
OSStatus  TXNGetFontMenuRef(TXNFontMenuObject iTXNFontMenuObject,MenuRef *oFontMenuHandle);
OSStatus  TXNDisposeFontMenuObject(TXNFontMenuObject iTXNFontMenuObject);
```

Selections

```
Boolean   TXNIsSelectionEmpty(TXNObject iTXNObject);
void      TXNGetSelection(TXNObject iTXNObject,TXNOffset *oStartOffset,TXNOffset
*oEndOffset);
void      TXNShowSelection(TXNObject iTXNObject,Boolean iShowEnd);
OSStatus  TXNSetSelection(TXNObject iTXNObject,TXNOffset iStartOffset,TXNOffset iEndOffset);
```

Setting Data

```
ByteCount TXNDataSize(TXNObject iTXNObject);
OSStatus  TXNGetData(TXNObject iTXNObject,TXNOffset iStartOffset,TXNOffset iEndOffset,
          Handle *oDataHandle);
OSStatus  TXNSetDataFromFile(TXNObject iTXNObject,SInt16 iFileRefNum,OSType iFileType,
          ByteCount iFileLength,TXNOffset iStartOffset,TXNOffset iEndOffset);
```

Getting the Change Count

```
ItemCount TXNGetChangeCount(TXNObject iTXNObject);
```

Scrap

```
Boolean   TXNIsScrapPastable(void);
OSStatus  TXNConvertToPublicScrap(void);
```

Font Defaults

```
OSStatus  TXNSetFontDefaults(TXNObject iTXNObject,ItemCount iCount,
          TXNMacOSPreferredFontDescription iFontDefaults[]);
OSStatus  TXNGetFontDefaults(TXNObject iTXNObject,ItemCount *ioCount,
          TXNMacOSPreferredFontDescription iFontDefaults[]);
```

Demonstration Program MLTETextEditor Listing

```
//
**********************************************************************************************
// MTLETextEditor.h                                                          CLASSIC EVENT
MODEL
//
**********************************************************************************************
//
// This program demonstrates the use of the Multilingual Text Engine API to create a basic
// multi-styled text editor. New documents created by the program are created and saved as
// Textension ('txtn') documents. Existing 'TEXT' documents and Unicode ('utxt') documents
// are saved in the original format. In the case of 'TEXT' documents, style information is
// saved in a 'styl' resource.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  An 'MBAR' resource, and 'MENU' resources for Apple, File, Edit, Size, Style, Color,
//    and Justification (preload, non-purgeable).
//
// •  A 'WIND' resource (purgeable) (initially not visible).
//
// •  A 'STR ' resource (purgeable) containing the "missing application name" string, which
is
//    copied to all document files created by the program.
//
// •  'STR#' resources (purgeable) containing error strings, the application's name (for
//    certain Navigation Services functions), and strings for the Edit menu Undo and Redo
//    items.
//
// •  A 'kind' resource (purgeable) describing file types, which is used by Navigation
//    Services to  build the native file types section of the Show pop-up menu in the Open
//    dialog box.
//
// •  An 'open' resource (purgeable) containing the file type list for the Open dialog box.
//
// •  The 'BNDL' resource (non-purgeable), 'FREF' resources (non-purgeable), signature
//    resource (non-purgeable), and icon family resources (purgeable), required to support
the
//    built application.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
//
**********************************************************************************************

// ..........................................................................
includes

#include <Carbon.h>

// ..........................................................................
defines

#define rMenubar                128
#define mAppleApplication       128
#define  iAbout                 1
#define mFile                   129
#define  iNew                   1
```

```
#define  iOpen                  2
#define  iClose                 4
#define  iSave                  5
#define  iSaveAs                6
#define  iRevert                7
#define  iPageSetup             9
#define  iPrint                 10
#define  iQuit                  12
#define mEdit                   130
#define  iUndo                  1
#define  iRedo                  2
#define  iCut                   4
#define  iCopy                  5
#define  iPaste                 6
#define  iClear                 7
#define  iSelectAll             8
#define mFont                   131
#define mSize                   132
#define  iTwelve                4
#define mStyle                  133
#define  iPlain                 1
#define  iBold                  3
#define  iUnderline             5
#define mColor                 134
#define  iBlack                 4
#define  iColorPicker          6
#define mJustification          135
#define  iDefault               1
#define  iLeft                  2
#define  iForceFull             6
#define mWindow                 136
#define mFirstHierarchical      160

#define rNewWindow              128
#define rAboutDialog            128
#define rErrorStrings           128
#define  eInstallHandler        1000
#define  eMaxWindows            1001
#define  eCantFindFinderProcess 1002
#define rMiscellaneousStrings   129
#define  sApplicationName       1
#define rOpenResource           128

#define kMaxWindows             8
#define kOpen                   0
#define kPrint                  1
#define kFileCreator            'bbJk'
#define MAX_UINT32              0xFFFFFFFF
#define MIN(a,b)                ((a) < (b) ? (a) : (b))
#define topLeft(r)              (((Point *) &(r))[0])

#define kATSUCGContextTag       32767L
// ..........................................................................function
prototypes

void      main                          (void);
void      doPreliminaries               (void);
void      doInitialiseMTLE              (void);
void      doInstallAEHandlers           (void);
void      eventLoop                     (void);
UInt32    doGetSleepTime                (void);
void      doIdle                        (void);
void      doEvents                      (EventRecord *);
void      doMouseDown                   (EventRecord *);
void      doBringFinderToFront          (void);
OSStatus  doFindProcess                 (OSType,OSType,ProcessSerialNumber *);
```

```
void      doActivate                  (EventRecord *);
void      doUpdate                    (EventRecord *);
Boolean   isApplicationWindow         (WindowRef,TXNObject *);
void      doAboutDialog               (void);
void      doSynchroniseFiles          (void);
OSStatus  openAppEventHandler         (AppleEvent *,AppleEvent *,SInt32);
OSStatus  reopenAppEventHandler       (AppleEvent *,AppleEvent *,SInt32);
OSStatus  openAndPrintDocsEventHandler (AppleEvent *,AppleEvent *,SInt32);
OSStatus  quitAppEventHandler         (AppleEvent *,AppleEvent *,SInt32);
OSStatus  doHasGotRequiredParams      (AppleEvent *);
void      doErrorAlert                (SInt16);
void      doCopyPString               (Str255,Str255);
void      doConcatPStrings            (Str255,Str255);

void      doEnableDisableMenus        (Boolean);
void      doAdjustAndPrepareMenus     (void);
void      doAdjustFileMenu            (MenuRef,WindowRef);
void      doAdjustEditMenu            (MenuRef,WindowRef);
void      doPrepareFontMenu           (WindowRef);
void      doPrepareSizeMenu           (MenuRef,WindowRef);
void      doPrepareStyleMenu          (MenuRef,WindowRef);
void      doPrepareColorMenu         (MenuRef,WindowRef);
Boolean   isEqualRGB                  (RGBColor *,RGBColor *);
void      doPrepareJustificationMenu  (MenuRef,WindowRef);
void      doMenuChoice                (SInt32);
void      doFileMenuChoice            (MenuItemIndex,WindowRef);
void      doEditMenuChoice            (MenuItemIndex,WindowRef);
void      doFontMenuChoice            (MenuID,MenuItemIndex,WindowRef);
void      doSizeMenuChoice            (MenuItemIndex,WindowRef);
void      doStyleMenuChoice           (MenuItemIndex,WindowRef);
void      doColorMenuChoice          (MenuItemIndex,WindowRef);
void      doJustificationMenuChoice   (MenuItemIndex,WindowRef);

OSStatus  doNewCommand                (void);
OSStatus  doOpenCommand               (void);
OSStatus  doCloseCommand              (NavAskSaveChangesAction);
OSStatus  doSaveCommand               (void);
OSStatus  doSaveAsCommand             (void);
OSStatus  doRevertCommand             (void);
OSStatus  doQuitCommand               (NavAskSaveChangesAction);
OSStatus  doNewDocWindow              (WindowRef *,FSSpec *,TXNFileType);
OSStatus  doOpenFile                  (FSSpec,OSType);
void      doCloseWindow               (WindowRef,TXNObject);
OSStatus  doWriteFile                 (WindowRef,Boolean);
OSStatus  doCopyResources             (FSSpec,TXNFileType,Boolean);
OSStatus  doCopyAResource             (ResType,SInt16,SInt16,SInt16);
void      navEventFunction            (NavEventCallbackMessage,NavCBRecPtr,
                                       NavCallBackUserData);

//
**********************************************************************************************
// MLTETextEditor.c
//
**********************************************************************************************

// ..........................................................................
includes

#include "MLTETextEditor.h"

// ...........................................................................global
variables

SInt16            gAppResFileRefNum;
Boolean           gRunningOnX = false;
Boolean           gDone;
```

```
TXNFontMenuObject gTXNFontMenuObject;
RgnHandle         gCursorRgnHdl;
extern SInt16     gCurrentNumberOfWindows;

// ****************************************************************************
main

void  main(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  OSStatus      osStatus = noErr;

  // ..................................................................do
preliminaries

  doPreliminaries();

  // ..................................................... save application's resource
file file reference number

  gAppResFileRefNum = CurResFile();

  // ..........................................................................
set up menu bar and menus

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    doErrorAlert(MemError());
  SetMenuBar(menubarHdl);

  CreateStandardWindowMenu(0,&menuRef);
  SetMenuID(menuRef,mWindow);
  InsertMenu(menuRef,0);
  DeleteMenuItem(menuRef,1);
  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
    }

    gRunningOnX = true;
  }

  // ..................................... build hierarchical font menu and draw menu
bar

  menuRef = GetMenuRef(mFont);
  osStatus = TXNNewFontMenuObject(menuRef,mFont,mFirstHierarchical,&gTXNFontMenuObject);
  if(osStatus != noErr)
    doErrorAlert(osStatus);

  DrawMenuBar();

  // ..................................................install required Apple event
handlers

  doInstallAEHandlers();

  // ..................................................................... enter event loop
```

```
  eventLoop();
}

// ************************************************************************
doPreliminaries

void  doPreliminaries(void)
{
  MoreMasterPointers(960);
  InitCursor();
  FlushEvents(everyEvent,0);

  doInitialiseMTLE();
}

// ************************************************************************
doInitializeMTLE

void  doInitialiseMTLE(void)
{
  TXNMacOSPreferredFontDescription  defaultFont[1];
  OSStatus                          osStatus = noErr;
  SInt16                            fontID;

  GetFNum("\pNew York",&fontID);

  defaultFont[0].fontID    = fontID;
  defaultFont[0].pointSize = 0x000C0000;
  defaultFont[0].fontStyle = kTXNDefaultFontStyle;
  defaultFont[0].encoding  = kTXNSystemDefaultEncoding;

  osStatus = TXNInitTextension(&defaultFont[0],1,kTXNWantMoviesMask);
  if(osStatus != noErr)
    doErrorAlert(osStatus);
}

// ************************************************************************
doInstallAEHandlers

void  doInstallAEHandlers(void)
{
  OSStatus osStatus = noErr;

  osStatus = AEInstallEventHandler(kCoreEventClass,kAEOpenApplication,
                         NewAEEventHandlerUPP((AEEventHandlerProcPtr) openAppEventHandler),
                         0L,false);
  if(osStatus != noErr)  doErrorAlert(eInstallHandler);

  osStatus = AEInstallEventHandler(kCoreEventClass,kAEReopenApplication,
                         NewAEEventHandlerUPP((AEEventHandlerProcPtr)
reopenAppEventHandler),
                         0L,false);
  if(osStatus != noErr)  doErrorAlert(eInstallHandler);

  osStatus = AEInstallEventHandler(kCoreEventClass,kAEOpenDocuments,
                 NewAEEventHandlerUPP((AEEventHandlerProcPtr)
openAndPrintDocsEventHandler),
                         kOpen,false);
  if(osStatus != noErr)  doErrorAlert(eInstallHandler);

  osStatus = AEInstallEventHandler(kCoreEventClass,kAEPrintDocuments,
                 NewAEEventHandlerUPP((AEEventHandlerProcPtr)
openAndPrintDocsEventHandler),
                         kPrint,false);
  if(osStatus != noErr)  doErrorAlert(eInstallHandler);
```

```
  osStatus = AEInstallEventHandler(kCoreEventClass,kAEQuitApplication,
                           NewAEEventHandlerUPP((AEEventHandlerProcPtr) quitAppEventHandler),
                           0L,false);
  if(osStatus != noErr)  doErrorAlert(eInstallHandler);
}

// ****************************************************************************
eventLoop

void  eventLoop(void)
{
  EventRecord eventStructure;

  gDone = false;
  gCursorRgnHdl = NewRgn();

  while(!gDone)
  {
    if(WaitNextEvent(everyEvent,&eventStructure,doGetSleepTime(),gCursorRgnHdl))
      doEvents(&eventStructure);
    else
    {
      if(eventStructure.what == nullEvent)
      {
        doIdle();
        doSynchroniseFiles();
      }
    }
  }
}

// ****************************************************************************
doGetSleepTime

UInt32  doGetSleepTime(void)
{
  WindowRef windowRef;
  UInt32    sleepTime;
  TXNObject txnObject = NULL;

  windowRef = FrontWindow();

  if(isApplicationWindow(windowRef,&txnObject))
    sleepTime = TXNGetSleepTicks(txnObject);
  else
    sleepTime = GetCaretTime();

  return sleepTime;
}

// ****************************************************************************
doIdle

void  doIdle(void)
{
  WindowRef windowRef;
  TXNObject txnObject = NULL;

  windowRef = FrontWindow();
  if(isApplicationWindow(windowRef,&txnObject))
  {
    if(TXNGetChangeCount(txnObject))
      SetWindowModified(windowRef,true);
  }
}
```

```
// ****************************************************************************
doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
  WindowRef windowRef;
  TXNObject txnObject = NULL;

  switch(eventStrucPtr->what)
  {
    case kHighLevelEvent:
      AEProcessAppleEvent(eventStrucPtr);
      break;
    case mouseDown:
      doMouseDown(eventStrucPtr);
      break;

    case keyDown:
      if(eventStrucPtr->modifiers & cmdKey)
      {
        doAdjustAndPrepareMenus();
        doMenuChoice(MenuEvent(eventStrucPtr));
      }
      break;

    case updateEvt:
      doUpdate(eventStrucPtr);
      break;

    case activateEvt:
      doActivate(eventStrucPtr);
      break;

    case osEvt:
      switch((eventStrucPtr->message >> 24) & 0x000000FF)
      {
        case suspendResumeMessage:
          if(eventStrucPtr->message & resumeFlag)
            SetThemeCursor(kThemeArrowCursor);
          break;

        case mouseMovedMessage:
          windowRef = FrontWindow();
          if(isApplicationWindow(windowRef,&txnObject))
            TXNAdjustCursor(txnObject,gCursorRgnHdl);
      }
      break;
  }
}

// ****************************************************************************
doMouseDown

void  doMouseDown(EventRecord *eventStrucPtr)
{
  WindowRef      windowRef;
  WindowPartCode partCode;
  OSStatus       osStatus  = noErr;
  TXNObject      txnObject = NULL;
  Boolean        handled    = false;
  SInt32         itemSelected;

  partCode = FindWindow(eventStrucPtr->where,&windowRef);

  switch(partCode)
  {
```

```
case inMenuBar:
  doAdjustAndPrepareMenus();
  doMenuChoice(MenuSelect(eventStrucPtr->where));
  break;
case inContent:
  if(windowRef != FrontWindow())
    SelectWindow(windowRef);
  else
  {
    if(isApplicationWindow(windowRef,&txnObject))
      TXNClick(txnObject,eventStrucPtr);
  }
  break;

case inGoAway:
  if(TrackGoAway(windowRef,eventStrucPtr->where))
    doCloseCommand(kNavSaveChangesClosingDocument);
  break;

case inProxyIcon:
  osStatus = TrackWindowProxyDrag(windowRef,eventStrucPtr->where);
  if(osStatus == errUserWantsToDragWindow)
    handled = false;
  else if(osStatus == noErr)
    handled = true;

case inDrag:
  if(!handled)
  {
    if(IsWindowPathSelectClick(windowRef,eventStrucPtr))
    {
      if(WindowPathSelect(windowRef,NULL,&itemSelected) == noErr)
      {
        if(LoWord(itemSelected) > 1)
          doBringFinderToFront();
      }

      handled = true;
    }
  }
  if(!handled)
    DragWindow(windowRef,eventStrucPtr->where,NULL);

  if(isApplicationWindow(windowRef,&txnObject))
    TXNAdjustCursor(txnObject,gCursorRgnHdl);

  break;

case inGrow:
  if(isApplicationWindow(windowRef,&txnObject))
  {
    TXNGrowWindow(txnObject,eventStrucPtr);
    TXNAdjustCursor(txnObject,gCursorRgnHdl);
  }
  break;

case inZoomIn:
case inZoomOut:
  if(TrackBox(windowRef,eventStrucPtr->where,partCode))
  {
    if(isApplicationWindow(windowRef,&txnObject))
    {
      TXNZoomWindow(txnObject,partCode);
      TXNAdjustCursor(txnObject,gCursorRgnHdl);
    }
  }
```

```
      break;
  }
}

// **************************************************************************
doBringFinderToFront

void  doBringFinderToFront(void)
{
  ProcessSerialNumber finderProcess;

  if(doFindProcess('MACS','FNDR',&finderProcess) == noErr)
    SetFrontProcess(&finderProcess);
  else
    doErrorAlert(eCantFindFinderProcess);
}

// **************************************************************************
doFindProcess

OSStatus  doFindProcess(OSType creator,OSType type,ProcessSerialNumber *outProcSerNo)
{
  ProcessSerialNumber procSerialNo;
  ProcessInfoRec      procInfoStruc;
  OSStatus            osStatus = noErr;

  procSerialNo.highLongOfPSN = 0;
  procSerialNo.lowLongOfPSN  = kNoProcess;

  procInfoStruc.processInfoLength = sizeof(ProcessInfoRec);
  procInfoStruc.processName       = NULL;
  procInfoStruc.processAppSpec    = NULL;
  procInfoStruc.processLocation   = NULL;

  while(true)
  {
    osStatus = GetNextProcess(&procSerialNo);
    if(osStatus != noErr)
      break;

    osStatus = GetProcessInformation(&procSerialNo,&procInfoStruc);
    if(osStatus != noErr)
      break;
    if((procInfoStruc.processSignature == creator) && (procInfoStruc.processType == type))
      break;
  }

  *outProcSerNo = procSerialNo;

  return osStatus;
}
// **************************************************************************
doActivate

void  doActivate(EventRecord *eventStrucPtr)
{
  WindowRef  windowRef;
  TXNObject  txnObject = NULL;
  Boolean    becomingActive;
  TXNFrameID txnFrameID = 0;

  windowRef = (WindowRef) eventStrucPtr->message;

  if(isApplicationWindow(windowRef,&txnObject))
  {
    becomingActive = ((eventStrucPtr->modifiers & activeFlag) == activeFlag);
```

```
    GetWindowProperty(windowRef,kFileCreator,'tFRM',sizeof(TXNFrameID),NULL,&txnFrameID);

    if(becomingActive)
      TXNActivate(txnObject,txnFrameID,becomingActive);
    else
      TXNActivate(txnObject,txnFrameID,becomingActive);

    TXNFocus(txnObject,becomingActive);
  }
}

// ****************************************************************************
doUpdate

void  doUpdate(EventRecord *eventStrucPtr)
{
  WindowRef windowRef;
  GrafPtr   oldPort;
  TXNObject txnObject = NULL;

  windowRef = (WindowRef) eventStrucPtr->message;

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

  if(isApplicationWindow(windowRef,&txnObject))
    TXNUpdate(txnObject);

  SetPort(oldPort);
}

// ************************************************************************
isApplicationWindow

Boolean  isApplicationWindow(WindowRef windowRef,TXNObject *txnObject)
{
  OSStatus osStatus = noErr;

  osStatus = GetWindowProperty(windowRef,kFileCreator,'tOBJ',sizeof(TXNObject),NULL,
                               txnObject);

  return (windowRef != NULL) && (GetWindowKind(windowRef) == kApplicationWindowKind);
}
// ************************************************************************
doAboutDialog

void  doAboutDialog(void)
{
  DialogRef dialogRef;
  SInt16    itemHit;

  dialogRef = GetNewDialog(rAboutDialog,NULL,(WindowRef) -1);
  ModalDialog(NULL,&itemHit);
  DisposeDialog(dialogRef);
}

// ************************************************************************
doSynchroniseFiles

void  doSynchroniseFiles(void)
{
  UInt32        currentTicks;
  WindowRef     windowRef;
  static UInt32 nextSynchTicks = 0;
  OSStatus      hasNoAliasHdl = noErr;
  Boolean       aliasChanged;
```

```
  AliasHandle   aliasHdl = NULL;
  FSSpec        newFSSpec;
  OSStatus      osStatus = noErr;
  SInt16        trashVRefNum;
  SInt32        trashDirID;
  TXNObject     txnObject = NULL;

  currentTicks = TickCount();
  windowRef    = FrontWindow();

  if(currentTicks > nextSynchTicks)
  {
    while(windowRef != NULL)
    {
      hasNoAliasHdl = GetWindowProperty(windowRef,kFileCreator,'tALH',sizeof(AliasHandle),
                                        NULL,&aliasHdl);
      if(hasNoAliasHdl)
        break;

      aliasChanged = false;
      ResolveAlias(NULL,aliasHdl,&newFSSpec,&aliasChanged);
      if(aliasChanged)
      {
        SetWindowProperty(windowRef,kFileCreator,'FiSp',sizeof(FSSpec),&newFSSpec);
        SetWTitle(windowRef,newFSSpec.name);
      }

      osStatus = FindFolder(kUserDomain,kTrashFolderType,kDontCreateFolder,
                            &trashVRefNum,&trashDirID);

      if(osStatus == noErr)
      {
        do
        {
          if(newFSSpec.parID == fsRtParID)
            break;

          if((newFSSpec.vRefNum == trashVRefNum) && (newFSSpec.parID == trashDirID))
          {
            GetWindowProperty(windowRef,kFileCreator,'tOBJ',sizeof(TXNObject),NULL,
                              &txnObject);
            TXNDeleteObject(txnObject);
            DisposeWindow(windowRef);
            gCurrentNumberOfWindows --;
            break;
          }
        }  while(FSMakeFSSpec(newFSSpec.vRefNum,newFSSpec.parID,"\p",&newFSSpec) == noErr);
      }

      windowRef = GetNextWindow(windowRef);
    }

    nextSynchTicks = currentTicks + 15;
  }
}

// ************************************************************************
openAppEventHandler

OSStatus  openAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefCon)
{
  OSStatus osStatus = noErr;

  osStatus = doHasGotRequiredParams(appEvent);
  if(osStatus == noErr)
    osStatus = doNewCommand();
```

```
  return osStatus;
}

// ********************************************************************
reopenAppEventHandler

OSStatus  reopenAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,
                                SInt32 handlerRefCon)
{
  OSStatus osStatus = noErr;

  osStatus = doHasGotRequiredParams(appEvent);
  if(osStatus == noErr)
    if(!FrontWindow())
      osStatus = doNewCommand();

  return osStatus;
}
// ************************************************************
openAndPrintDocsEventHandler

OSStatus  openAndPrintDocsEventHandler(AppleEvent *appEvent,AppleEvent *reply,
                                       SInt32 handlerRefcon)
{
  FSSpec     fileSpec;
  AEDescList docList;
  OSStatus   osStatus, ignoreErr;
  SInt32     index, numberOfItems;
  Size       actualSize;
  AEKeyword  keyWord;
  DescType   returnedType;
  FInfo      fileInfo;
  TXNObject  txnObject;

  osStatus = AEGetParamDesc(appEvent,keyDirectObject,typeAEList,&docList);

  if(osStatus == noErr)
  {
    osStatus = doHasGotRequiredParams(appEvent);
    if(osStatus == noErr)
    {
      osStatus = AECountItems(&docList,&numberOfItems);
      if(osStatus == noErr)
      {
        for(index=1;index<=numberOfItems;index++)
        {
          osStatus = AEGetNthPtr(&docList,index,typeFSS,&keyWord,&returnedType,
                                 &fileSpec,sizeof(fileSpec),&actualSize);
          if(osStatus == noErr)
          {
            osStatus = FSpGetFInfo(&fileSpec,&fileInfo);
            if(osStatus == noErr)
            {
              if(osStatus = doOpenFile(fileSpec,fileInfo.fdType))
                doErrorAlert(osStatus);

              if(osStatus == noErr && handlerRefcon == kPrint)
              {
                if(isApplicationWindow(FrontWindow(),&txnObject))
                {
                  if(osStatus = TXNPrint(txnObject))
                    doErrorAlert(osStatus);

                  if(osStatus = doCloseCommand(kNavSaveChangesOther))
                    doErrorAlert(osStatus);
```

```
              }
            }
          }
        }
        else
          doErrorAlert(osStatus);
      }
    }
  }
  else
    doErrorAlert(osStatus);

  ignoreErr = AEDisposeDesc(&docList);
 }
 else
  doErrorAlert(osStatus);

 return osStatus;
}

// **********************************************************************
quitAppEventHandler

OSStatus  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
 OSStatus osStatus = noErr;

 osStatus = doHasGotRequiredParams(appEvent);
 if(osStatus == noErr)
 {
  while(FrontWindow())
  {
   osStatus = doCloseCommand(kNavSaveChangesQuittingApplication);

   if(osStatus != noErr && osStatus != kNavAskSaveChangesCancel)
    doErrorAlert(osStatus);
   if(osStatus == kNavAskSaveChangesCancel)
    return noErr;
  }
 }

 gDone = true;

 return osStatus;
}

// **********************************************************************
doHasGotRequiredParams

OSStatus  doHasGotRequiredParams(AppleEvent *appEvent)
{
 DescType returnedType;
 Size     actualSize;
 OSStatus osStatus = noErr;

 osStatus = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,
                              NULL,0,&actualSize);
 if(osStatus == errAEDescNotFound)
  osStatus = noErr;
 else if(osStatus == noErr)
  osStatus = errAEParamMissed;

 return osStatus;
}
// **********************************************************************
doErrorAlert
```

```
void  doErrorAlert(SInt16 errorCode)
{
  Str255 errorString, theString;
  SInt16 itemHit;

  if(errorCode == kATSUFontsMatched)
    return;

  if(errorCode == eInstallHandler)
    GetIndString(errorString,rErrorStrings,1);
  else if(errorCode == eMaxWindows)
    GetIndString(errorString,rErrorStrings,2);
  else if(errorCode == eCantFindFinderProcess)
    GetIndString(errorString,rErrorStrings,3);
  else
  {
    GetIndString(errorString,rErrorStrings,4);
    NumToString((SInt32) errorCode,theString);
    doConcatPStrings(errorString,theString);
  }

  if(errorCode != memFullErr)
    StandardAlert(kAlertCautionAlert,errorString,NULL,NULL,&itemHit);
  else
  {
    StandardAlert(kAlertStopAlert,errorString,NULL,NULL,&itemHit);
    ExitToShell();
  }
}

// ************************************************************************
doCopyPString

void  doCopyPString(Str255 sourceString,Str255 destinationString)
{
  SInt16 stringLength;

  stringLength = sourceString[0];
  BlockMove(sourceString + 1,destinationString + 1,stringLength);
  destinationString[0] = stringLength;
}

// ************************************************************************
doConcatPStrings

void  doConcatPStrings(Str255 targetString,Str255 appendString)
{
  SInt16 appendLength;

  appendLength = MIN(appendString[0],255 - targetString[0]);

  if(appendLength > 0)
  {
    BlockMoveData(appendString+1,targetString+targetString[0]+1,(SInt32) appendLength);
    targetString[0] += appendLength;
  }
}
//
****************************************************************************************
// MLTEMenus.c
//
****************************************************************************************

// ........................................................................
includes
```

```
#include "MLTETextEditor.h"

// ..................................................................... global
variables

RGBColor                gCurrentColorPickerColor = {  0x0000,0x0000,0x0000 } ;
extern SInt16           gCurrentNumberOfWindows;
extern TXNFontMenuObject gTXNFontMenuObject;
extern Boolean          gDone;

// ********************************************************************
doEnableDisableMenus

void  doEnableDisableMenus(Boolean enableMenus)
{
  if(enableMenus)
  {
    EnableMenuItem(GetMenuRef(mEdit),0);
    EnableMenuItem(GetMenuRef(mFont),0);
    EnableMenuItem(GetMenuRef(mSize),0);
    EnableMenuItem(GetMenuRef(mStyle),0);
    EnableMenuItem(GetMenuRef(mColor),0);
    EnableMenuItem(GetMenuRef(mJustification),0);
    EnableMenuItem(GetMenuRef(mWindow),0);
  }
  else
  {
    DisableMenuItem(GetMenuRef(mEdit),0);
    DisableMenuItem(GetMenuRef(mFont),0);
    DisableMenuItem(GetMenuRef(mSize),0);
    DisableMenuItem(GetMenuRef(mStyle),0);
    DisableMenuItem(GetMenuRef(mColor),0);
    DisableMenuItem(GetMenuRef(mJustification),0);
    DisableMenuItem(GetMenuRef(mWindow),0);
  }
}

// ********************************************************************
doAdjustAndPrepareMenus

void  doAdjustAndPrepareMenus(void)
{
  WindowRef windowRef;

  windowRef = FrontWindow();

  doAdjustFileMenu(GetMenuRef(mFile),windowRef);
  doAdjustEditMenu(GetMenuRef(mEdit),windowRef);
  doPrepareFontMenu(windowRef);
  doPrepareSizeMenu(GetMenuRef(mSize),windowRef);
  doPrepareStyleMenu(GetMenuRef(mStyle),windowRef);
  doPrepareColorMenu(GetMenuRef(mColor),windowRef);
  doPrepareJustificationMenu(GetMenuRef(mJustification),windowRef);
  DrawMenuBar();
}

// ********************************************************************
doAdjustFileMenu

void  doAdjustFileMenu(MenuRef menuRef,WindowRef windowRef)
{
  TXNObject txnObject = NULL;

  if(gCurrentNumberOfWindows <= kMaxWindows)
  {
```

```
    EnableMenuItem(menuRef,iNew);
    EnableMenuItem(menuRef,iOpen);
  }
  else
  {
    DisableMenuItem(menuRef,iNew);
    DisableMenuItem(menuRef,iOpen);
  }

  if(isApplicationWindow(windowRef,&txnObject))
  {
    EnableMenuItem(menuRef,iClose);

    if(TXNGetChangeCount(txnObject))
    {
      EnableMenuItem(menuRef,iSave);
      EnableMenuItem(menuRef,iRevert);
    }
    else
    {
      DisableMenuItem(menuRef,iSave);
      DisableMenuItem(menuRef,iRevert);
    }

    if(TXNDataSize(txnObject))
    {
      EnableMenuItem(menuRef,iSaveAs);
      EnableMenuItem(menuRef,iPageSetup);
      EnableMenuItem(menuRef,iPrint);
    }
    else
    {
      DisableMenuItem(menuRef,iSaveAs);
      DisableMenuItem(menuRef,iPageSetup);
      DisableMenuItem(menuRef,iPrint);
    }
  }
  else
  {
    DisableMenuItem(menuRef,iClose);
    DisableMenuItem(menuRef,iSave);
    DisableMenuItem(menuRef,iSaveAs);
    DisableMenuItem(menuRef,iRevert);
    DisableMenuItem(menuRef,iPageSetup);
    DisableMenuItem(menuRef,iPrint);
  }
}

// ************************************************************************
doAdjustEditMenu

void  doAdjustEditMenu(MenuRef menuRef,WindowRef windowRef)
{
  TXNObject    txnObject = NULL;
  SInt16       menuItem;
  Str255       itemText;
  TXNActionKey actionKey;

  if(isApplicationWindow(windowRef,&txnObject))
  {
    // ..................................................................... disable all
items

    for(menuItem = iUndo;menuItem <= iSelectAll;menuItem ++)
      DisableMenuItem(menuRef,menuItem);
```

```
    // ..................................... undo and redo default - can't undo, can't
redo

    GetIndString(itemText,130,1);
    SetMenuItemText(menuRef,iUndo,itemText);
    GetIndString(itemText,130,2);
    SetMenuItemText(menuRef,iRedo,itemText);

    // ..................................... if undoable, enable undo item and set item
text

    if(TXNCanUndo(txnObject,&actionKey))
    {
      EnableMenuItem(menuRef,iUndo);

      if((actionKey < kTXNTypingAction) || (actionKey > kTXNMoveAction))
        actionKey = -1;

      GetIndString(itemText,130,2 * actionKey + 5);
        SetMenuItemText(menuRef,iUndo,itemText);
    }

    // ..................................... if redoable, enable redo item and set
item text

    if(TXNCanRedo(txnObject,&actionKey))
    {
      EnableMenuItem(menuRef,iRedo);

      if((actionKey < kTXNTypingAction) || (actionKey > kTXNMoveAction))
        actionKey = -1;

      GetIndString(itemText,130,2 * actionKey + 6);
      SetMenuItemText(menuRef,iRedo,itemText);
    }
    // ..................................... if there is a selection, enable cut,
copy, and clear

    if(!TXNIsSelectionEmpty(txnObject))
    {
      EnableMenuItem(menuRef,iCut);
      EnableMenuItem(menuRef,iCopy);
      EnableMenuItem(menuRef,iClear);
    }

    // ............................                  ..........if scrap is pastable,
enable paste

    if(TXNIsScrapPastable())
      EnableMenuItem(menuRef,iPaste);

    // ..................... if any characters in TXNObject, enable select all

    if(TXNDataSize(txnObject))
      EnableMenuItem(menuRef,iSelectAll);
  }
}

// ************************************************************************
doPrepareFontMenu

void  doPrepareFontMenu(WindowRef windowRef)
{
  TXNObject  txnObject = NULL;

  if(isApplicationWindow(windowRef,&txnObject))
```

```
    TXNPrepareFontMenu(txnObject,gTXNFontMenuObject);
}

// ************************************************************************
doPrepareSizeMenu

void  doPrepareSizeMenu(MenuRef menuRef,WindowRef windowRef)
{
  TXNObject          txnObject = NULL;
  static Fixed       itemSizes[8] = {  0x00090000,0x000A0000,0x000B0000,0x000C0000,
                                       0x000E0000,0x00120000,0x00180000,0x00240000 } ;
  TXNContinuousFlags txnContinuousFlags = 0;
  TXNTypeAttributes  txnTypeAttributes;
  OSStatus           osStatus = noErr;
  SInt16             menuItem;

  if(isApplicationWindow(windowRef,&txnObject))
  {
    txnTypeAttributes.tag            = kTXNQDFontSizeAttribute;
    txnTypeAttributes.size           = kTXNFontSizeAttributeSize;
    txnTypeAttributes.data.dataValue = 0;

    osStatus = TXNGetContinuousTypeAttributes(txnObject,&txnContinuousFlags,1,
                                              &txnTypeAttributes);
    if(osStatus == noErr)
    {
      for(menuItem = 1;menuItem < 8;menuItem ++)
      {
        CheckMenuItem(menuRef,menuItem,(txnContinuousFlags & kTXNSizeContinuousMask) &&
                      (txnTypeAttributes.data.dataValue == itemSizes[menuItem - 1]));
      }
    }
  }
}

// ************************************************************************
doPrepareStyleMenu

void  doPrepareStyleMenu(MenuRef menuRef,WindowRef windowRef)
{
  TXNObject          txnObject = NULL;
  TXNContinuousFlags txnContinuousFlags = 0;
  TXNTypeAttributes  txnTypeAttributes;
  OSStatus           osStatus = noErr;
  SInt16             menuItem;

  if(isApplicationWindow(windowRef,&txnObject))
  {
    txnTypeAttributes.tag            = kTXNQDFontStyleAttribute;
    txnTypeAttributes.size           = kTXNQDFontStyleAttributeSize;
    txnTypeAttributes.data.dataValue = 0;

    osStatus = TXNGetContinuousTypeAttributes(txnObject,&txnContinuousFlags,1,
                                              &txnTypeAttributes);
    if(osStatus == noErr)
    {
      CheckMenuItem(menuRef,iPlain,(txnContinuousFlags & kTXNStyleContinuousMask) &&
                    (txnTypeAttributes.data.dataValue == normal));

      for(menuItem = iBold;menuItem <= iUnderline;menuItem ++)
      {
        CheckMenuItem(menuRef,menuItem,(txnContinuousFlags & kTXNStyleContinuousMask) &&
                      (txnTypeAttributes.data.dataValue & (1 << (menuItem - iBold))));
      }
    }
  }
```

```
}

// ************************************************************************
doPrepareColorMenu

void  doPrepareColorMenu(MenuRef menuRef,WindowRef windowRef)
{
  TXNObject          txnObject = NULL;
  TXNContinuousFlags txnContinuousFlags = 0;
  TXNTypeAttributes  txnTypeAttributes;
  RGBColor           attributesColor;
  OSStatus           osStatus = noErr;
  SInt16             menuItem;
  RGBColor           itemColors[4] = { { 0xFFFF,0x0000,0x0000 } ,{  0x0000,0x8888,0x0000 } ,
                                       { 0x0000,0x0000,0xFFFF } ,
{  0x0000,0x0000,0x0000 }  } ;
  if(isApplicationWindow(windowRef,&txnObject))
  {
    txnTypeAttributes.tag          = kTXNQDFontColorAttribute;
    txnTypeAttributes.size         = kTXNQDFontColorAttributeSize;
    txnTypeAttributes.data.dataPtr = &attributesColor;

    osStatus = TXNGetContinuousTypeAttributes(txnObject,&txnContinuousFlags,1,
                                              &txnTypeAttributes);
    if(osStatus == noErr)
    {
      for(menuItem = 1;menuItem < 5;menuItem ++)
      {
        CheckMenuItem(menuRef,menuItem,(txnContinuousFlags & kTXNColorContinuousMask) &&
                      (isEqualRGB(&attributesColor,&itemColors[menuItem - 1])));
      }
    }
  }
}

// ************************************************************************
isEqualRGB

Boolean  isEqualRGB(RGBColor *attributesColor,RGBColor *itemColor)
{
  return (attributesColor->red   == itemColor->red &&
          attributesColor->green == itemColor->green &&
          attributesColor->blue  == itemColor->blue);
}

// ************************************************************
doPrepareJustificationMenu

void  doPrepareJustificationMenu (MenuRef menuRef,WindowRef windowRef)
{
  TXNObject      txnObject = NULL;
  static UInt32  itemJustifications[6] = {  kTXNFlushDefault,kTXNFlushLeft,kTXNFlushRight,
                                            kTXNCenter,kTXNFullJust,kTXNForceFullJust} ;
  TXNControlTag  txnControlTag[1];
  TXNControlData txnControlData[1];
  SInt16         menuItem;
  OSStatus       osStatus = noErr;

  if(isApplicationWindow(windowRef,&txnObject))
  {
    txnControlTag[0]         = kTXNJustificationTag ;
    txnControlData[0].uValue = 0;

    osStatus = TXNGetTXNObjectControls(txnObject,1,txnControlTag,txnControlData);

    if(osStatus == noErr)
```

```
    {
      for(menuItem = iDefault;menuItem <= iForceFull;menuItem ++ )
        CheckMenuItem(menuRef,menuItem,(txnControlData[0].uValue ==
                      itemJustifications[menuItem - 1]));
    }
  }
}
// ****************************************************************************
doMenuChoice

void  doMenuChoice(SInt32 menuChoice)
{
  MenuID        menuID;
  MenuItemIndex menuItem;
  OSStatus      osStatus = noErr;
  WindowRef     windowRef;
  TXNObject     txnObject = NULL;

  windowRef = FrontWindow();

  menuID = HiWord(menuChoice);
  menuItem = LoWord(menuChoice);

  if(menuID == 0)
    return;

  switch(menuID)
  {
    case mAppleApplication:
      if(menuItem == iAbout)
        doAboutDialog();
      break;

    case mFile:
      doFileMenuChoice(menuItem,windowRef);
      break;

    case mEdit:
      doEditMenuChoice(menuItem,windowRef);
      break;

    case mFont:
      doFontMenuChoice(menuID,menuItem,windowRef);
      break;

    case mSize:
      doSizeMenuChoice(menuItem,windowRef);
      break;

    case mStyle:
      doStyleMenuChoice(menuItem,windowRef);
      break;

    case mColor:
      doColorMenuChoice(menuItem,windowRef);
      break;

    case mJustification:
      doJustificationMenuChoice(menuItem,windowRef);
      break;
    default:
      if(menuID >= mFirstHierarchical)
        doFontMenuChoice(menuID,menuItem,windowRef);
      break;
  }
```

```
  HiliteMenu(0);
}

// ********************************************************************
doFileMenuChoice

void  doFileMenuChoice(MenuItemIndex menuItem,WindowRef windowRef)
{
  TXNObject txnObject = NULL;
  OSStatus  osStatus = noErr;

  switch(menuItem)
  {
    case iNew:
      if(osStatus = doNewCommand())
        doErrorAlert(osStatus);
      break;

    case iOpen:
      if(osStatus = doOpenCommand())
        doErrorAlert(osStatus);
      break;

    case iClose:
      if((osStatus = doCloseCommand(kNavSaveChangesClosingDocument)) &&
          osStatus != kNavAskSaveChangesCancel)
        doErrorAlert(osStatus);
      break;

    case iSave:
      if(osStatus = doSaveCommand())
        doErrorAlert(osStatus);
      break;

    case iSaveAs:
      if(osStatus = doSaveAsCommand())
        doErrorAlert(osStatus);
      break;

    case iRevert:
      if(osStatus = doRevertCommand())
        doErrorAlert(osStatus);
      break;

    case iPageSetup:
      if(isApplicationWindow(windowRef,&txnObject))
      {
        osStatus = TXNPageSetup(txnObject);
        if(osStatus != userCanceledErr && osStatus != noErr)
          doErrorAlert(osStatus);
      }
      break;

    case iPrint:
      if(isApplicationWindow(windowRef,&txnObject))
      {
        osStatus = TXNPrint(txnObject);
        if(osStatus != userCanceledErr && osStatus != noErr)
          doErrorAlert(osStatus);
      }
      break;

    case iQuit:
      if((osStatus = doQuitCommand(kNavSaveChangesQuittingApplication)) &&
          osStatus != kNavAskSaveChangesCancel)
        doErrorAlert(osStatus);
```

```
      if(osStatus != kNavAskSaveChangesCancel)
      {
        if(gTXNFontMenuObject != NULL)
        {
          if(osStatus = TXNDisposeFontMenuObject(gTXNFontMenuObject))
            doErrorAlert(osStatus);
        }

        gTXNFontMenuObject = NULL;

        TXNTerminateTextension();
        gDone = true;
      }
      break;
  }
}

// ****************************************************************************
doEditMenuChoice

void  doEditMenuChoice(MenuItemIndex menuItem,WindowRef windowRef)
{
  TXNObject txnObject = NULL;
  OSStatus  osStatus = noErr;

  if(isApplicationWindow(windowRef,&txnObject))
  {
    switch(menuItem)
    {
      case iUndo:
        TXNUndo(txnObject);
        break;

      case iRedo:
        TXNRedo(txnObject);
        break;
      case iCut:
        if((osStatus = TXNCut(txnObject)) == noErr)
          TXNConvertToPublicScrap();
        else
          doErrorAlert(osStatus);
        break;

      case iCopy:
        if((osStatus = TXNCopy(txnObject)) == noErr)
          TXNConvertToPublicScrap();
        else
          doErrorAlert(osStatus);
        break;

      case iPaste:
        if(osStatus = TXNPaste(txnObject))
          doErrorAlert(osStatus);
        break;

      case iClear:
        if(osStatus = TXNClear(txnObject))
          doErrorAlert(osStatus);
        break;

      case iSelectAll:
        TXNSelectAll(txnObject);
        break;
    }
  }
```

```
}

// ************************************************************************
doFontMenuChoice

void  doFontMenuChoice(MenuID menuID,MenuItemIndex menuItem,WindowRef windowRef)
{
  TXNObject txnObject = NULL;
  OSStatus  osStatus = noErr;

  if(isApplicationWindow(windowRef,&txnObject))
  {
    if(gTXNFontMenuObject != NULL)
    {
      if(osStatus = TXNDoFontMenuSelection(txnObject,gTXNFontMenuObject,menuID,menuItem))
        doErrorAlert(osStatus);
    }
  }
}
// ************************************************************************
doSizeMenuChoice

void  doSizeMenuChoice(MenuItemIndex menuItem,WindowRef windowRef)
{
  TXNObject         txnObject = NULL;
  static Fixed      itemSizes[8] = {  0x00090000,0x000A0000,0x000B0000,0x000C0000,
                                      0x000E0000,0x00120000,0x00180000,0x00240000 } ;
  Fixed             sizeToSet;
  TXNTypeAttributes txnTypeAttributes;
  OSStatus          osStatus = noErr;

  if(isApplicationWindow(windowRef,&txnObject))
  {
    sizeToSet = itemSizes[menuItem - 1];

    txnTypeAttributes.tag            = kTXNQDFontSizeAttribute;
    txnTypeAttributes.size           = kTXNFontSizeAttributeSize;
    txnTypeAttributes.data.dataValue = sizeToSet;

    if(TXNSetTypeAttributes(txnObject,1,&txnTypeAttributes,kTXNUseCurrentSelection,
                            kTXNUseCurrentSelection))
      doErrorAlert(osStatus);
  }
}

// ************************************************************************
doStyleMenuChoice

void  doStyleMenuChoice(MenuItemIndex menuItem,WindowRef windowRef)
{
  TXNObject         txnObject = NULL;
  static Style      itemStyles[5] = {  normal,0,bold,italic,underline } ;
  Style             styleToSet;
  TXNTypeAttributes txnTypeAttributes;
  OSStatus          osStatus = noErr;

  if(isApplicationWindow(windowRef,&txnObject))
  {
    styleToSet = itemStyles[menuItem - 1];

    txnTypeAttributes.tag            = kTXNQDFontStyleAttribute;
    txnTypeAttributes.size           = kTXNQDFontStyleAttributeSize;
    txnTypeAttributes.data.dataValue = styleToSet;

    if(TXNSetTypeAttributes(txnObject,1,&txnTypeAttributes,kTXNUseCurrentSelection,
                            kTXNUseCurrentSelection))
```

```
      doErrorAlert(osStatus);
  }
}
// ************************************************************************
doColorMenuChoice

void  doColorMenuChoice(MenuItemIndex menuItem,WindowRef windowRef)
{
  TXNObject         txnObject = NULL;
  Point             where;
  Boolean           colorPickerButton;
  Str255            prompt = "\pPick a text color";
  RGBColor          colorToSet;
  RGBColor          itemColors[4] = {  {  0xFFFF,0x0000,0x0000 } ,
{  0x0000,0x8888,0x0000 } ,
                                       {  0x0000,0x0000,0xFFFF } ,
{  0x0000,0x0000,0x0000 }  } ;
  TXNTypeAttributes txnTypeAttributes;
  OSStatus          osStatus = noErr;

  if(isApplicationWindow(windowRef,&txnObject))
  {
    if(menuItem == iColorPicker)
    {
      where.v = where.h = 0;
      colorPickerButton = GetColor(where,prompt,&gCurrentColorPickerColor,&colorToSet);
      if(colorPickerButton)
        gCurrentColorPickerColor = colorToSet;
      else
        return;
    }
    else
      colorToSet = itemColors[menuItem - 1];

    txnTypeAttributes.tag          = kTXNQDFontColorAttribute;
    txnTypeAttributes.size         = kTXNQDFontColorAttributeSize;
    txnTypeAttributes.data.dataPtr = &colorToSet;

    if(TXNSetTypeAttributes(txnObject,1,&txnTypeAttributes,kTXNUseCurrentSelection,
                            kTXNUseCurrentSelection))
      doErrorAlert(osStatus);
  }
}

// ************************************************************
doJustificationMenuChoice

void  doJustificationMenuChoice(MenuItemIndex menuItem,WindowRef windowRef)
{
  TXNObject      txnObject = NULL;
  static UInt32  itemJustifications[6] = {  kTXNFlushDefault,kTXNFlushLeft,kTXNFlushRight,
                                            kTXNCenter,kTXNFullJust,kTXNForceFullJust } ;
  OSStatus       osStatus = noErr;
  UInt32         justificationToSet;
  TXNControlTag  txnControlTag[1];
  TXNControlData txnControlData[1];

  if(isApplicationWindow(windowRef,&txnObject))
  {
    justificationToSet = itemJustifications[menuItem - 1];
    txnControlTag[0] = kTXNJustificationTag;

    osStatus = TXNGetTXNObjectControls(txnObject,1,txnControlTag,txnControlData);

    if(txnControlData[0].uValue != justificationToSet)
    {
```

```
      txnControlData[0].uValue = justificationToSet;
      osStatus = TXNSetTXNObjectControls(txnObject,false,1,txnControlTag,txnControlData);
      if(osStatus != noErr)
        doErrorAlert(osStatus);
    }
  }
}

//
****************************************************************************************
// MLTENewOpenCloseSave.c
//
****************************************************************************************

// ..........................................................................
includes

#include "MLTETextEditor.h"

// ..................................................................... global
variables

SInt16 gCurrentNumberOfWindows = 0;
SInt16 gUntitledWindowNumber = 0;

extern Boolean gRunningOnX = false;
extern SInt16  gAppResFileRefNum;

// ************************************************************************
doNewCommand

OSStatus  doNewCommand(void)
{
  OSStatus  osStatus = noErr;
  WindowRef windowRef;

  if(gCurrentNumberOfWindows == kMaxWindows)
    return eMaxWindows;

  osStatus = doNewDocWindow(&windowRef,NULL,kTXNTextensionFile);

  if(osStatus == noErr)
    SetWindowProxyCreatorAndType(windowRef,kFileCreator,kTXNTextensionFile,kUserDomain);

  return osStatus;
}
// ************************************************************************
doOpenCommand

OSStatus doOpenCommand(void)
{
  OSStatus          osStatus = noErr;
  NavDialogOptions  dialogOptions;
  NavTypeListHandle fileTypeListHdl = NULL;
  NavEventUPP       navEventFunctionUPP;
  NavReplyRecord    navReplyStruc;
  SInt32            count, index;
  AEKeyword         theKeyword;
  DescType          actualType;
  FSSpec            fileSpec;
  Size              actualSize;
  FInfo             fileInfo;
  OSType            fileType;

  osStatus = NavGetDefaultDialogOptions(&dialogOptions);
```

```
  if(osStatus == noErr)
  {
    GetIndString(dialogOptions.clientName,rMiscellaneousStrings,sApplicationName);
    fileTypeListHdl = (NavTypeListHandle) GetResource('open',rOpenResource);

    navEventFunctionUPP = NewNavEventUPP((NavEventProcPtr) navEventFunction);

    osStatus = NavGetFile(NULL,&navReplyStruc,&dialogOptions,navEventFunctionUPP,
                          NULL,NULL,fileTypeListHdl,NULL);

    DisposeNavEventUPP(navEventFunctionUPP);

    if(osStatus == noErr && navReplyStruc.validRecord)
    {
      osStatus = AECountItems(&(navReplyStruc.selection),&count);
      if(osStatus == noErr)
      {
        for(index=1;index<=count;index++)
        {
          osStatus = AEGetNthPtr(&(navReplyStruc.selection),index,typeFSS,&theKeyword,
                                 &actualType,&fileSpec,sizeof(fileSpec),&actualSize);

          if((osStatus = FSpGetFInfo(&fileSpec,&fileInfo)) == noErr)
          {
            fileType = fileInfo.fdType;
            osStatus = doOpenFile(fileSpec,fileType);
          }
        }
      }

      osStatus = NavDisposeReply(&navReplyStruc);
    }
    if(fileTypeListHdl != NULL)
      ReleaseResource((Handle) fileTypeListHdl);
  }

  if(osStatus == userCanceledErr)
    osStatus = noErr;

  return osStatus;
}

// ********************************************************************************
doCloseCommand

OSStatus  doCloseCommand(NavAskSaveChangesAction action)
{
  WindowRef               windowRef;
  TXNObject               txnObject = NULL;
  OSStatus                osStatus = noErr;
  NavDialogOptions        dialogOptions;
  NavAskSaveChangesResult reply = 0;
  NavEventUPP             navEventFunctionUPP;
  Str255                  fileName;

  osStatus = NavGetDefaultDialogOptions(&dialogOptions);

  if(osStatus == noErr)
  {
    windowRef = FrontWindow();
    if(isApplicationWindow(windowRef,&txnObject))
    {
      if(TXNGetChangeCount(txnObject))
      {
        GetWTitle(windowRef,fileName);
        BlockMoveData(fileName,dialogOptions.savedFileName,fileName[0] + 1);
```

```
        GetIndString(dialogOptions.clientName,rMiscellaneousStrings,sApplicationName);

        navEventFunctionUPP = NewNavEventUPP((NavEventProcPtr) navEventFunction);

        osStatus = NavAskSaveChanges(&dialogOptions,action,&reply,navEventFunctionUPP,0);

        DisposeNavEventUPP(navEventFunctionUPP);

        if(osStatus == noErr)
        {
          switch(reply)
          {
            case kNavAskSaveChangesSave:
              if((osStatus = doSaveCommand()) == noErr)
                doCloseWindow(windowRef,txnObject);
              break;

            case kNavAskSaveChangesDontSave:
                doCloseWindow(windowRef,txnObject);
              break;
            case kNavAskSaveChangesCancel:
              osStatus = kNavAskSaveChangesCancel;
              break;
          }
        }
      }
      else
      {
        doCloseWindow(windowRef,txnObject);
      }
    }
  }

  return osStatus;
}

// ***************************************************************************
doSaveCommand

OSStatus  doSaveCommand(void)
{
  WindowRef windowRef;
  OSStatus  hasNoFileSpec;
  OSStatus  osStatus = noErr;
  FSSpec    fileSpec;

  windowRef = FrontWindow();

  hasNoFileSpec = GetWindowProperty(windowRef,kFileCreator,'FiSp',sizeof(FSSpec),NULL,
                                    &fileSpec);
  if(hasNoFileSpec)
    osStatus = doSaveAsCommand();
  else
    osStatus = doWriteFile(windowRef,false);

  if(osStatus == noErr)
    SetWindowModified(windowRef,false);

  return osStatus;
}

// ***************************************************************************
doSaveAsCommand

OSStatus  doSaveAsCommand(void)
{
```

```
OSStatus         osStatus = noErr;
NavDialogOptions dialogOptions;
WindowRef        windowRef;
NavEventUPP      navEventFunctionUPP;
TXNFileType      txnFileType;
NavReplyRecord   navReplyStruc;
AEKeyword        theKeyword;
DescType         actualType;
FSSpec           fileSpec;
Size             actualSize;
AliasHandle      aliasHdl;
osStatus = NavGetDefaultDialogOptions(&dialogOptions);

if(osStatus == noErr)
{
  windowRef = FrontWindow();

  GetWTitle(windowRef,dialogOptions.savedFileName);
  GetIndString(dialogOptions.clientName,rMiscellaneousStrings,sApplicationName);

  navEventFunctionUPP = NewNavEventUPP((NavEventProcPtr) navEventFunction);

  GetWindowProperty(windowRef,kFileCreator,'FiTy',sizeof(TXNFileType),NULL,&txnFileType);

  osStatus = NavPutFile(NULL,&navReplyStruc,&dialogOptions,navEventFunctionUPP,
                        txnFileType,kFileCreator,NULL);

  DisposeNavEventUPP(navEventFunctionUPP);

  if(navReplyStruc.validRecord && osStatus == noErr)
  {
    if((osStatus = AEGetNthPtr(&(navReplyStruc.selection),1,typeFSS,&theKeyword,
                               &actualType,&fileSpec,sizeof(fileSpec),&actualSize))
                               == noErr)
    {
      if(!navReplyStruc.replacing)
      {
        osStatus = FSpCreate(&fileSpec,kFileCreator,txnFileType,navReplyStruc.keyScript);
        if(osStatus != noErr)
        {
          NavDisposeReply(&navReplyStruc);
          return osStatus;
        }
      }

      if(osStatus == noErr)
      {
        SetWTitle(windowRef,fileSpec.name);

        SetWindowProperty(windowRef,kFileCreator,'FiSp',sizeof(FSSpec),&fileSpec);
        SetPortWindowPort(windowRef);
        SetWindowProxyFSSpec(windowRef,&fileSpec);
        GetWindowProxyAlias(windowRef,&aliasHdl);
        SetWindowProperty(windowRef,kFileCreator,'tALH',sizeof(AliasHandle),&aliasHdl);
        SetWindowModified(windowRef,false);

        osStatus = doWriteFile(windowRef,!navReplyStruc.replacing);
      }

      NavCompleteSave(&navReplyStruc,kNavTranslateInPlace);
    }

    NavDisposeReply(&navReplyStruc);
  }
}
if(osStatus == userCanceledErr)
```

```
    osStatus = noErr;

  return osStatus;
}

// ************************************************************************
doRevertCommand

OSStatus  doRevertCommand(void)
{
  OSStatus               osStatus = noErr;
  NavDialogOptions       dialogOptions;
  NavEventUPP            navEventFunctionUPP;
  WindowRef              windowRef;
  Str255                 fileName;
  NavAskSaveChangesResult reply;
  TXNObject              txnObject = NULL;

  osStatus = NavGetDefaultDialogOptions(&dialogOptions);

  if(osStatus == noErr)
  {
    navEventFunctionUPP = NewNavEventUPP((NavEventProcPtr) navEventFunction);

    windowRef = FrontWindow();
    GetWTitle(windowRef,fileName);
    BlockMoveData(fileName,dialogOptions.savedFileName,fileName[0] + 1);

    osStatus = NavAskDiscardChanges(&dialogOptions,&reply,navEventFunctionUPP,0);

    DisposeNavEventUPP(navEventFunctionUPP);

    if(osStatus == noErr)
    {
      if(reply == kNavAskDiscardChanges)
      {
        if(isApplicationWindow(windowRef,&txnObject))
        {
          TXNRevert(txnObject);

          if(TXNDataSize(txnObject))
            SetWindowModified(windowRef,false);
        }
      }
    }
  }

  return osStatus;
}
// ************************************************************************
doQuitCommand

OSStatus  doQuitCommand(NavAskSaveChangesAction action)
{
  OSStatus osStatus = noErr;

  while(FrontWindow())
  {
    osStatus = doCloseCommand(action);
    if(osStatus != noErr)
      return osStatus;
  }

  return osStatus;
}
```

```
// ****************************************************************************
doNewDocWindow

OSStatus  doNewDocWindow(WindowRef *outWindowRef,FSSpec *fileSpec,TXNFileType txnFileType)
{
  WindowRef       windowRef;
  Str255          numberAsString, titleString = "\puntitled ";
  Rect            availableBoundsRect, portRect;
  SInt16          windowHeight;
  TXNFrameOptions txnFrameOptions;
  OSStatus        osStatus  = noErr;
  TXNObject       txnObject  = NULL;
  TXNFrameID      txnFrameID;
  RGBColor        frameColor = {  0xEEEE, 0xEEEE, 0xEEEE } ;
  TXNControlTag   txnControlTag[1];
  TXNControlData  txnControlData[1];
  TXNMargins      txnMargins;
  CGContextRef    cgContextRef;

  // ..................................................................... get
window

  if(!(windowRef = GetNewCWindow(rNewWindow,NULL,(WindowRef) -1)))
    return MemError();
  SetPortWindowPort(windowRef);

  ChangeWindowAttributes(windowRef,kWindowInWindowMenuAttribute,0);

  gUntitledWindowNumber++;
  if(gUntitledWindowNumber != 1)
  {
    NumToString(gUntitledWindowNumber,numberAsString);
    doConcatPStrings(titleString,numberAsString);
  }
  SetWTitle(windowRef,titleString);
  // ............................... extend window bottom to bottom of screen less the
dock

  GetAvailableWindowPositioningBounds(GetMainDevice(),&availableBoundsRect);
  GetWindowPortBounds(windowRef,&portRect);
  LocalToGlobal(&topLeft(portRect));
  windowHeight = availableBoundsRect.bottom - portRect.top;
  SizeWindow(windowRef,630,windowHeight,false);

  // ........................................... get new TXNObject and attach window to
it

  txnFrameOptions = kTXNWantHScrollBarMask | kTXNWantVScrollBarMask | kTXNShowWindowMask;

  osStatus = TXNNewObject(fileSpec,windowRef,NULL,txnFrameOptions,
kTXNTextEditStyleFrameType,
                          txnFileType,kTXNSystemDefaultEncoding,&txnObject,&txnFrameID,
                          NULL);

  if(osStatus == noErr)
  {
    // ..................................... associate frame ID and TXNObject with
window

    SetWindowProperty(windowRef,kFileCreator,'tOBJ',sizeof(TXNObject),&txnObject);
    SetWindowProperty(windowRef,kFileCreator,'tFRM',sizeof(TXNFrameID),&txnFrameID);
    if(fileSpec != NULL)
      SetWindowProperty(windowRef,kFileCreator,'FiSp',sizeof(FSSpec),fileSpec);
    SetWindowProperty(windowRef,kFileCreator,'FiTy',sizeof(TXNFileType),&txnFileType);

    // ..................................................................... set
```

```
margins

    txnControlTag[0] = kTXNMarginsTag;
    txnControlData[0].marginsPtr = &txnMargins;

    txnMargins.leftMargin  = txnMargins.topMargin = 10;
    txnMargins.rightMargin = txnMargins.bottomMargin = 10;
    TXNSetTXNObjectControls(txnObject,false,1,txnControlTag,txnControlData);

    // ....................................create core graphics context and pass to MLTE

    if(gRunningOnX)
    {
      CreateCGContextForPort(GetWindowPort(windowRef),&cgContextRef);
      txnControlTag[0] = kATSUCGContextTag;
      txnControlData[0].uValue = (UInt32) cgContextRef;
      TXNSetTXNObjectControls(txnObject,false,1,txnControlTag,txnControlData);
    }
  }
  else
    doErrorAlert(osStatus);

  gCurrentNumberOfWindows ++;
  if(gCurrentNumberOfWindows == 1)
    doEnableDisableMenus(true);

  *outWindowRef = windowRef;

  return noErr;
}
// ********************************************************************************
doOpenFile

OSStatus  doOpenFile(FSSpec fileSpec,OSType fileType)
{
  OSStatus    osStatus = noErr;
  WindowRef   windowRef;
  AliasHandle aliasHdl;

  if(osStatus = doNewDocWindow(&windowRef,&fileSpec,fileType))
    return osStatus;

  SetWTitle(windowRef,fileSpec.name);

  SetWindowProxyFSSpec(windowRef,&fileSpec);
  GetWindowProxyAlias(windowRef,&aliasHdl);
  SetWindowProperty(windowRef,kFileCreator,'tALH',sizeof(AliasHandle),&aliasHdl);

  SetWindowModified(windowRef,false);

  return noErr;
}

// ********************************************************************************
doCloseFile

void  doCloseWindow(WindowRef windowRef,TXNObject txnObject)
{
  TXNDeleteObject(txnObject);
  DisposeWindow(windowRef);
  gCurrentNumberOfWindows --;
  if(gCurrentNumberOfWindows == 0)
    doEnableDisableMenus(false);
}

// ********************************************************************************
```

doWriteFile

```
OSStatus  doWriteFile(WindowRef windowRef,Boolean newFile)
{
  TXNPermanentTextEncodingType encodingType;
  TXNObject   txnObject = NULL;
  FSSpec      fileSpec, fileSpecTemp;
  TXNFileType txnFileType;
  UInt32      currentTime;
  Str255      tempFileName;
  OSStatus    osStatus  = noErr;
  SInt16      tempFileVolNum, tempFileRefNum, tempResForkRefNum = -1;
  SInt32      tempFileDirID;
  Boolean     hasResFile = false;

  GetWindowProperty(windowRef,kFileCreator,'tOBJ',sizeof(TXNObject),NULL,&txnObject);
  GetWindowProperty(windowRef,kFileCreator,'FiSp',sizeof(FSSpec),NULL,&fileSpec);
  GetWindowProperty(windowRef,kFileCreator,'FiTy',sizeof(TXNFileType),NULL,&txnFileType);

  encodingType = (txnFileType == kTXNTextFile) ? kTXNMacOSEncoding : kTXNUnicodeEncoding;
  GetDateTime(&currentTime);
  NumToString((SInt32) currentTime,tempFileName);

  osStatus = FindFolder(fileSpec.vRefNum,kTemporaryFolderType,kCreateFolder,&tempFileVolNum,
                        &tempFileDirID);
  if(osStatus == noErr)
    osStatus = FSMakeFSSpec(tempFileVolNum,tempFileDirID,tempFileName,&fileSpecTemp);
  if(osStatus == noErr || osStatus == fnfErr)
    osStatus = FSpCreate(&fileSpecTemp,'trsh','trsh',smSystemScript);
  if(osStatus == noErr)
    osStatus = FSpOpenDF(&fileSpecTemp,fsRdWrPerm,&tempFileRefNum);

  if(osStatus == noErr)
  {
    if(txnFileType == kTXNTextFile)
    {
      FSpCreateResFile(&fileSpecTemp,'trsh','trsh',smSystemScript);
      osStatus = ResError();
      if(osStatus == noErr)
        tempResForkRefNum = FSpOpenResFile(&fileSpecTemp,fsRdWrPerm);
      hasResFile = true;
    }
  }

  if(osStatus == noErr)
    osStatus = TXNSave(txnObject,txnFileType,kTXNMultipleStylesPerTextDocumentResType,
                       encodingType,&fileSpec,tempFileRefNum,tempResForkRefNum);

  if(osStatus == noErr)
    osStatus = FSpExchangeFiles(&fileSpecTemp,&fileSpec);
  if(osStatus == noErr)
    osStatus = FSpDelete(&fileSpecTemp);

  if(osStatus == noErr)
    osStatus = FSClose(tempFileRefNum);
  if(osStatus == noErr)
    if(tempResForkRefNum != -1)
      CloseResFile(tempResForkRefNum);

  osStatus = ResError();

  if(osStatus == noErr)
    if(newFile)
      osStatus = doCopyResources(fileSpec,txnFileType,hasResFile);

  return osStatus;
```

```
}
// ************************************************************************
doCopyResources

OSStatus  doCopyResources(FSSpec fileSpec,TXNFileType fileType,Boolean hasResFile)
{
  OSStatus osStatus = noErr;
  SInt16   fileRefNum;

  if(!hasResFile)
    FSpCreateResFile(&fileSpec,kFileCreator,fileType,smSystemScript);

  osStatus = ResError();
  if(osStatus == noErr)
    fileRefNum = FSpOpenResFile(&fileSpec,fsRdWrPerm);

  if(fileRefNum > 0)
    osStatus = doCopyAResource('STR ',-16396,gAppResFileRefNum,fileRefNum);
  else
    osStatus = ResError();

  if(osStatus == noErr)
    CloseResFile(fileRefNum);

  osStatus = ResError();

  return osStatus;
}

// ************************************************************************
doCopyAResource

OSStatus  doCopyAResource(ResType resourceType,SInt16 resourceID,SInt16 sourceFileRefNum,
                      SInt16 destFileRefNum)
{
  Handle  sourceResourceHdl;
  Str255  sourceResourceName;
  ResType ignoredType;
  SInt16  ignoredID;

  UseResFile(sourceFileRefNum);

  sourceResourceHdl = GetResource(resourceType,resourceID);

  if(sourceResourceHdl != NULL)
  {
    GetResInfo(sourceResourceHdl,&ignoredID,&ignoredType,sourceResourceName);
    DetachResource(sourceResourceHdl);
    UseResFile(destFileRefNum);
    AddResource(sourceResourceHdl,resourceType,resourceID,sourceResourceName);
    if(ResError() == noErr)
      UpdateResFile(destFileRefNum);
  }

  ReleaseResource(sourceResourceHdl);

  return ResError();
}
// ************************************************************************
navEventFunction

void  navEventFunction(NavEventCallbackMessage callBackSelector,NavCBRecPtr callBackParms,
                       NavCallBackUserData callBackUD)
{
  WindowRef windowRef;
```

```
  switch(callBackSelector)
  {
    case kNavCBEvent:
      switch(callBackParms->eventData.eventDataParms.event->what)
      {
        case updateEvt:
          windowRef = (WindowRef) callBackParms->eventData.eventDataParms.event->message;
          if(GetWindowKind(windowRef) != kDialogWindowKind)
            doUpdate((EventRecord *) callBackParms->eventData.eventDataParms.event);
          break;
      }
      break;
  }
}

//
*****************************************************************************************
```

Demonstration Program MLTETextEditor Comments

This program, like all previous programs, demonstrates the programming of one particular aspect of the Mac OS. However, unlike all previous demonstration programs, it can also be used as a useful application, that is, as a fully functional basic text editor.

New documents created by the text editor are created and saved as Textension (`'txtn'`) file types. Existing files of type `'TEXT'` and Unicode (`'utxt'`) can be opened are saved as `'TEXT'` and Unicode files. For `'TEXT'` files, style information is saved in `'styl'` resources. Movies may be embedded within documents.

Those areas of the program relating to file operations and window proxy icons follow the same general approach as does the demonstration program files (see Chapter 18). This includes the file synchronization function and the functions for copying the missing application name string resource to the resource fork of saved files. The Apple event handlers are identical to those in the demonstration program files, except for the added capability to handle the Print Documents event.

MLTETextEditor.c

Main

The application's resource fork file reference number is saved for use in the function doCopyResources.

`CreateStandardWindowMenu` is used to create a Window menu, which is then given an ID and added to the menu list. If the program is running on Mac OS 8/9, the first item in the Window menu (Zoom Window) is deleted. (As will be seen in this program, `TXNZoomWindow` is called when the user clicks the zoom box/button. `TXNZoomWindow` adjusts the scroll bars automatically; however, this does not occur when Zoom Window is chosen from a Window menu. It is thus necessary to delete the item in this particular program.)

After a reference to the Font menu is obtained, `TXNNewFontMenuObject` is called to create a hierarchical Font menu. Note that the value passed in the third parameter, which specifies the ID of the first sub-menu, must be 160 or higher.

doInitializeMLTE

`doInitializeMLTE` is called from `doPreliminaries`. The call to

`TXNInitTextension` initializes the Textension library. Font information specifying that the default font, font size, and font style for the system default encoding (Unicode on systems with ATSUI) be New York, 12 point, normal is passed in the first parameter. The encoding field specifies how the application wants to see text. The third parameter specifies that embedded movies are to be supported.

doInstallAEHandlers

Note that `openAndPrintDocsEventHandler` will be called when both an Open Documents and a Print Documents event is received, the difference being that the reference constant is set to `kOpen` (0) for an Open Application event and to `kPrint` (1) for a Print Documents event.

eventLoop

Note that the value passed in `WaitNextEvent`'s sleep parameter is the value returned by a call to the function `doGetSleepTime`. Note also that, when a NULL event is returned by `WaitNextEvent`, the functions doIdle and `doSynchroniseFiles` are called.

doGetSleepTime

`doGetSleepTime` determines the value passed in `WaitNextEvent`'s sleep parameter.

This is the first of many functions that call the function `isApplicationWindow`. As will be seen, `isApplicationWindow` returns true if there is an open window and if it is of the application kind. It also returns, in the `txnObject` parameter, the `TXNObject` to which the window was attached when the window was created.

If there is an open window, and if it is of the application kind, `TXNGetSleepTicks` is called to get the sleep time to be passed to `WaitNextEvent`. This ensures that the function `doIdle` will be called at the appropriate interval. If the front window is of the dialog kind, the sleep time is set to the value returned by a call to `GetCaretTime`. (Actually, in this application, which presents no dialogs with edit text items, it might be considered more appropriate to set the sleep time to the maximum unsigned long value at the else statement.)

doIdle

`doIdle` is called to perform idle processing.

The call to `TXNGetChangeCount` gets the number of times the document has been changed since the last time the `TXNObject` was saved. If any changes have been made since the last save, `SetWindowModified` is called to disable the window proxy icon.

doEvents

At the keyDown case, note that `TXNKeyDown` does not need to be called in a Carbon application. Carbon special-cases Command-key events to avoid them being sent to MLTE. However, all other key-downs get sent directly to the Type Services Manager and your application never gets to "see" them. This means that, in Carbon applications, you cannot filter out characters for special handling before they are passed to MLTE (TXNKeyDown) as you could in a Classic application.

The only exception is Command-key events. For command keys, because MLTE has the habit of eating all keystrokes that go to it, even command keys that it can't process, we detect if the command key exists in the menus and special-case it to avoid sending it to MLTE.

At the `mouseMovedMessage` case within the osEvt case, `TXNAdjustCursor` is called to handle cursor shape changing. If the mouse is over a text area, `TXNAdjustCursor` sets the cursor to the I-beam cursor. If the cursor is over a movie, over a scroll bar, or outside a text area, `TXNAdjustCursor` sets the cursor to the arrow cursor.

doMouseDown

At the inGrow case, `TXNGrowWindow` is called to handle the resizing operation. At the `inZoomIn/inZoomOut` case, `TXNZoomWindow` is called to zoom the window. At the `inDrag`, `inGrow`, and `inZoomIn/inZoomOut` cases, `TXNAdjustCursor` is called after the window has been dragged, re-sized, or zoomed so that the mouse-moved region is re-calculated.

doActivate

As will be seen, when `TXNObject` is created and a window attached to it, `SetWindowProperty` is called to associate the `TXNObject` frame ID with the window. The call to `GetWindowProperty` retrieves this frame ID.

If the window is becoming active, `TXNActivate` is called, with true passed in the third parameter, to activate the scroll bars. Also, `TXNFocus` is called, with true passed in the second parameter to activate text input (selection and typing). If the window is becoming inactive, false is passed in `TXNActivate`'s third parameter and `TXNFocus`' second parameter to deactivate the scroll bars and text input.

doUpdate

`TXNUpdate` is called to redraw everything in the content area. Note that this function calls `BeginUpdate` and `EndUpdate`, so it is not necessary for the application to do so.

isApplicationWindow

`isApplicationWindow` is called from many functions. It returns `true` if there is a front window and if that window is of the application kind. It also returns to the caller the `TXNObject` to which that window is attached. As will be seen, the `TXNObject` is associated with the window when both are created, and is retrieved here by the call to `GetWindowProperty`.

doSynchroniseFiles

`doSynchroniseFiles` is the file synchronization function (see Chapter 18). It is adapted from the function of the same name in the demonstration program files. In this version

- The method used to determine whether the window has a file associated with it is to call `GetWindowProperty` in an attempt to retrieve the handle to the alias structure which, as will be seen, is associated with a window by a call to `SetWindowProperty` when a file is saved or loaded.
- If the `aliasChanged` parameter is set to `true` in the call to `ResolveAlias`, meaning that the location of the file has changed, the file system specification structure returned by `ResolveAlias` is associated with the window by the call to `SetWindowProperty`, replacing the previous file system specification structure stored in the window.
- At the inner `if` statement, if the file is found to be in the trash, `GetWindowProperty` is called to return the `TXNObject` associated with the window when it was created, `TXNDeleteObject` is called to delete the `TXNObject` and its associated data structures, and DisposeWindow is called to dispose of the window.

openAndPrintDocsEventHandler

`openAndPrintDocsEventHandler` is called when an Open Documents or Print Documents Apple event is received. In both cases, `doOpenFile` is called to open and display the file. In the case of a Print Documents event, `TXNPrint` is also called to print the document, following which `doCloseCommand` is called to dispose of the window and its `TXNObject`.

doErrorAlert

If the error code is `kATSUFontsMatched` (-8793), `doErrorAlert` simply returns. `kATSUFontsMatched` is not an error as such, but is returned by `ATSUMatchFontsToText` when changes need to be made to the fonts associated with the text.

MLTEMenus.c

doEnableDisableMenus

`doEnableDisableMenus` is called from `doCloseWindow` and `doNewDocWindow` to ensure that all menus except the File menu are disabled if no windows are open and that those menus are enabled if at least one window is open.

doAdjustFileMenu

If the call to `TXNGetChangeCount` reveals that the document has been changed since it was opened or last saved, the File menu Save and Revert items are enabled, otherwise they are disabled.

If the call to `TXNDataSize` reveals that there are characters in the `TXNObject`, the Save As, Page Setup, and Print items are enabled, otherwise they are disabled.

The else block executes only if no windows are open, ensuring that all File menu items except New, Open, and Quit are disabled.

doEditMenu

At the first block, all Edit menu items are disabled. At the second block, the default item text for both the Undo and Redo items (Can't Undo, Can't Redo) is set. This may be changed by the next two blocks.

The next block addresses the Undo item. The call to `TXNCanUndo` determines whether the last action is undoable. If the last action is undoable, `TXNCanUndo` returns, in the second parameter, an action key code that will be used to index a STR# resource for a string describing the undoable action. If this action key code represents a typing, cut, paste, clear, change font, change font color, change font size, change font style, change alignment, drag action, or move action, the appropriate string is retrieved by the call to `GetIndString` and the item text is set to this string (for example, Undo Cut). If the action is any other action, the item text is set to Undo.

At the block beginning with the call to `TXNCanRedo`, the same process is repeated in respect of the Redo item.

If the call to `TXNIsSelectionEmpty` reveals that the current selection is not empty, the Cut, Copy, and Clear items are enabled.

If the call to `TXNIsScrapPastable` reveals that the current scrap contains data that is supported by MLTE, the Paste item is enabled.

If the call to `TXNDataSize` reveals that there are characters in the `TXNObject`, the Select All item is enabled.

doPrepareFontMenu

`doAdjustFileMenu` and `doAdjustEditMenu` are concerned with enabling and disabling menu items as appropriate. `doPrepareFontMenu` and the other menu preparation functions are concerned with adding and removing checkmarks from items.

For the Font menu, all that is required is a call to `TXNPrepareFontMenu`. If the insertion point caret is in text in a particular font, or if a selection contains text in a single font, that menu item will be checkmarked. (If the font is in a Font menu sub-menu, the item in the sub-menu is checkmarked and a "dash" marking character is placed in the Font menu item to which the sub-menu is attached.) On the other hand, if a selection contains text in more than one font, all marking characters are removed from the Font menu and its sub-menus.

doPrepareSizeMenu

`doPrepareSizeMenu` does for the Size menu what `doPrepareFontMenu` does for the Font menu. If the insertion point caret is in text in a particular size, or if a selection contains text in a single size, the associated Size menu item is checkmarked. On the other hand, if a selection contains text in more than one size, all Size menu items are un-checkmarked.

Font size is represented by a value of type Fixed (four bytes comprising 16-bit signed integer plus 16-bit fraction). Accordingly, the `itemSizes` array is initialized with the sizes represented in the Size menu (9, 10, 11, 12, 14, 18, 24, 36) expressed as Fixed values. Each element in the array corresponds to an individual menu item.

The tag and size fields of a structure of type `TXNTypeAttributes` are assigned, respectively, a value ('size') specifying the size attribute and the size of the Fixed data type. The call to `TXNGetContinuousTypeAttributes` tests the current selection to see if the font size is continuous. On output, bit 1 in the second parameter (`txContinuousFlags`) will be set if all the text in the selection is all of one size, and the dataValue field of the data field of `txnTypeAttributes` will contain that size.

All items in the menu are then walked. If bit 1 of `txContinuousFlags` is not set, all items will be un-checkmarked. If bit 1 is set, and if the font size returned in the dataValue field is equal to the value in that element of the `itemSizes` array corresponding to the current menu item, that item is checkmarked, otherwise it is uncheckmarked.

doPrepareStyleMenu

The same general approach is used to prepare the Style menu. In this case, the tag and size fields of a structure of type `TXNTypeAttributes` are assigned, respectively, a value ('face') specifying the style attribute and the size of the Style data type. Also, the block that checkmarks or un-checkmarks the menu items is a little different, reflecting the fact that the bold, italic, and underline styles can be cumulative.

The first call to `CheckMenuItem` checkmarks the Plain menu item if all the text in the selection is of the same style and if that style is the plain (normal) style. The for loop addresses the bold, italic, and underline menu items only. If all the text in the selection is of the same style, or combination of styles, the menu item/s corresponding to the bits set in the dataValue field of the data field of `txnTypeAttributes` are checkmarked, otherwise, they are uncheckmarked.

doPrepareColorMenu

`doPrepareColorMenu` is similar to `doAdjustSizeMenu` except that the tag and size fields of a structure of type `TXNTypeAttributes` are assigned, respectively, a value ('klor') specifying the color attribute and the size of the RGBColor data type. Note also that the address of the `attributesColor` variable is assigned to the `dataPtr` field of the data field of `txnTypeAttributes`, meaning that `attributesColor` receives the color returned by the call to `TXNGetContinuousTypeAttributes`. It is the color stored in `attributesColor` that is compared with the colors stored in the `itemColors` array in order to determine whether a `menuItem` should be checked or unchecked (assuming that the selection contains text in one color only).

doPrepareJustificationMenu

In `doPrepareJustificationMenu`, the first element of a single-element array of type `TXNControl` is assigned the control tag '`just`'. The call to `TXNGetTXNObjectControls` returns, in the `uValue` field of the first element of a single-element array of type `TXNControlData`, a value representing the current justification setting in the `TXNObject`. This value determines which item in the Justification menu is checkmarked, all other items being uncheckmarked.

doFileMenuChoice

`doFileMenuChoice` is broadly similar to the function of the same name in the demonstration program Files, except as follows.

At the `iPageSetup` case, `TXNPageSetup` is called. `TXNPageSetup` displays the Page Setup dialog and handles all text re-formatting arising from user interaction with the dialog.

At the `iPrint` case, `TXNPrint` is called. `TXNPrint` displays the Print dialog and prints

the document.

At the `iQuit` case, if close-down has not been interrupted by the user clicking in the Cancel button of a Save Changes dialog, `TXNDisposeFontMenuObject` is called to dispose of the `TXNFontMenuObject` created at program start. Note that, even if the object is successfully disposed of, it is still necessary to set the associated global variable to `NULL`.

doEditMenuChoice

At the `iUndo` and `iRedo` cases, `TXNUndo` and `TXNRedo` are called to undo and redo the last action.

At the `iCut` and `iCopy` cases, `TXNCut` and `TXNCopy` are called to cut and copy the current selection to MLTE's private scrap. `TXNConvertToPublicScrap` is also called to copy MLTE's private scrap to the public scrap (clipboard). Note that, for reasons explained in Chapter 20, `TXNConvertToPublicScrap` must not be called at a suspend event in a Carbon application.

At the `iPaste` case, `TXNPaste` is called to paste MLTE's private scrap to the document. Note that there is no need to precede this call with a call to `TXNConvertFromPublicScrap` in a Carbon application. In a Carbon application, MLTE keeps the public scrap (clipboard) synchronized with MLTE's private scrap.

At the i`Clea`r case, `TXNClear` is called to delete the current selection without copying it to the MLTE private scrap. At the `iSelectAll` case, `TXNSelectAll` is called to select everything in the frame.

doFontMenuChoice

`doFontMenuChoice` handles choices from the Font menu. The call to `TXNFontMenuSelection` takes a menu ID and menu item index and changes the current selection to the font represented by that menu item.

doSizeMenuChoice

`doSizeMenuChoice` handles choices from the Size menu. The received menu item index is used to determine which element of the `itemSizes` array is assigned to the variable sizeToSet. The tag and size fields of a structure of type `TXNTypeAttributes` are then assigned, respectively, a value ('`size`') specifying the size attribute and the size of the Fixed data type. The `dataValue` field of the data field is assigned the size to set. The call to `TXNSetTypeAttributes` sets the font size in the specified `TXNObject`.

doStyleMenuChoice and doColorMenuChoice

`doStyleMenuChoice` and `doColorMenuChoice` handle choices from the Style and Color menus, and use the same general approach as `doSizeMenuChoice`. The exception is that, in `doColorMenuChoice`, if the Color Picker item is chosen, `GetColor` is called to present the Color Picker dialog to solicit a color choice by the user.

doJustificationMenuChoice

`doJustificationMenuChoice` handles choices from the Justification menu. The received menu item index is used to determine which element of the `itemJustification` array is assigned to the variable `justificationToSet`. The single element of an array of type `TXNControlTag` is then assigned a value ('`just`') specifying the justification tag. The call to `TXNGetTXNObjectControls` returns, in the fourth parameter, the `TXNObject`'s current justification setting. If this setting is not the same as the justification the user is attempting to set, `TXNSetTXNObjectControls` is called to set the chosen justification in the `TXNObject` false is passed in the second parameter so that all controls are not reset to the defaults.

MLTENewOpenCloseSave.c

The file handling functions in this section are broadly similar to those in the demonstration program files. This includes those areas of the code relating to window proxy icons. Generally, only the code that differs from the files code is explained in the following.

Fig 1 shows the general File menu and Apple event handling strategy, as adapted from Fig 4 in Chapter 18.

doNewCommand

`doNewCommand` is called when the user chooses New from the File menu, and from the Open Application and Re-Open Application Apple Event handlers.

If the call to `doNewDocWindow`, a reference to the created window will be returned in the first parameter, `NULL` is passed in the second (file system specification) parameter, and the third parameter specifies the required file type as Textension. (Note: If you prefer the file type for documents created by the program to be TEXT or Unicode, the only actions required are to pass `kTXNTextFile` or `kTXNUnicodeTextFile` in the third parameter of the calls to `doNewDocWindow` and `SetWindowProxyCreatorAndType`.)

doOpenCommand

`doOpenCommand` is called when the user chooses Open from the File menu. Recall that the aim is to get the file system specification and file type for the file, or files, selected in

the Navigation Services Open dialog and pass them in a call to `doOpenFile`.

doCloseCommand

`doCloseCommand` is called when the user chooses Close from the File menu, when the user clicks the go-away box of a window, and for each open window when the user chooses Quit from the File menu or the Quit Application Apple event handler is invoked.

if the call to `TXNGetChangeCount` reveals that no changes have been made to the document since it was opened, or since the last save, `doCloseWindow` is called. If changes have been made, a Navigation Services Save Changes dialog box is presented. If the user clicks the Save button, `doSaveCommand` and then `doCloseWindow` are called. If the user clicks the Don't Save button, `doCloseWindow` is called. If the user clicks the Cancel button, that fact is simply reported to the calling function and no other action is taken.

doSaveCommand

`doSaveCommand` is called when the user chooses Save from the File menu and by `doCloseCommand` if the user clicks the Save button in the Save Changes dialog box.

As will be seen, a file system specification is only associated with a window when an existing document is opened or a new document is saved. Thus the call to `GetWindowProperty` will return `false` if a new document has not yet been saved. In this case, `doSaveAsCommand` is called to solicit a filename from the user and then save the document to that file by calling `doWriteFile`. If a file system specification is already associated with the window, the call to `doSaveAsCommand` is bypassed and `doWriteFile` is called to save the file to the existing filename.

doSaveAsCommand

`doSaveAsCommand` is called when the user chooses Save As from the File menu and from `doSaveCommand` when the front window does not yet have a file associated with it.

As will be seen, the file type is associated with the window when the window is created. The call to `GetWindowProperty` retrieves the file type so that it can be passed in the fifth parameter of the call to `NavPutFile` and, if the file is not being replaced, in the third parameter of the call to `FSpCreate`.

The two calls to `SetWindowProperty` associates the file system specification returned by `AEGetNthPtr`, and a handle to the alias data for the file returned by the call to `GetWindowProxyAlias`, with the window. (The latter is used by the file synchronization function.)

The call to `doWriteFile` writes the file.

doRevertCommand

If, when the Discard Changes dialog box is presented, the user clicks the OK button, `TXNRevert` is called to revert to the last saved version of the document or, if the file was not previously saved, to revert to an empty document. The call to `TXNDataSize` determines whether the revert has been to an empty document. If not, `SetWindowModified` is called with false passed in the modified parameter to cause the window proxy icon to appear in the enabled state, indicating no unsaved changes.

doQuitCommand

`doQuitCommand` is called when the user chooses Quit from the File menu. For each open window, `doCloseCommand` is called.

doNewDocWindow

`doNewDocWindow` creates a new window, creates a new `TXNObject` and attaches the window to it, associates certain information with the window, sets the backgound to a light grey color (for demonstration purposes), and sets the margins.

`GetNewCWindow` creates a new invisible window and `SetPortWindowPort` makes its graphics port the current port. The call to `ChangeWindowAttributes` ensures that the window's title will appear as an item in the Window menu.

The next block adjusts the height of the (invisible) window so that the bottom is just above the space occupied by the Mac OS 8/9 control strip.

Preparatory to the call to `TXNNewObject`, a variable of type `TXNFrameOptions` is assigned a value that will specify that the created `TXNObject` is to support horizontal and vertical scroll bars and that the window should be displayed before the call to `TXNNewObject` returns.

The call to `TXNNewObject` creates a new `TXNObject` and attaches it to the window specified in the second parameter. `NULL` is passed in the third (iFrame) parameter, meaning that the window's port rectangle will be used as the frame. Note that `kTXNTextensionFile` will be passed in the sixth (`iFileType`) parameter if a new document is being created, and that `kTXNTextFile` or `kTXNUnicodeTextFile` will be passed in if a '`TEXT`' or Unicode file is being opened. Note also that the local variable `txnFrameID` will contain the frame ID when `TXNNewObject` returns.

If a pointer to file system specification structure is passed in `TXNNewObject`'s first parameter, `TXNNewObject` will read in the file and display its contents. If `NULL` is

passed in this parameter, the document will start empty. (Recall that `NULL` will be received in the `fileSpec` formal parameter when `doNewDocWindow` is called from `doNewCommand`, and a pointer to a file system specification structure will be received when `doNewDocWindow` is called from `doOpenFile`.)

The next block associates the `TXNObject`, the frame ID, and the received file system specification (if any) and file type with the window.

The next block sets the margins to 10 pixels all round. The first element of a single-element array of type `TXNControlTag` is assigned a value ('`marg`') specifying the margins tag, and the `marginsPtr` field in the first element a single-element array of type `TXNControlData` is assigned the address of a local variable of type `TXNMargins`. The four fields of the `TXNMargins` structure are then assigned the value 10. The call to `TXNSetTXNObjectControls` sets the margins.

By default, MLTE renders text via QuickDraw on Mac OS X. The appearance of text on Mac OS X is greatly enhanced by rendering via Core Graphics. Accordingly, a Core Graphics context is created and the information is passed to MLTE by calling `TXNSetTXNObjectControls` with the `kATSUCGContextTag`.

If the window and `TXNObject` were successfully created, the global variable that keeps track of the number of open windows is incremented. If the previous number of open windows was zero, meaning that all of the applications menus less the Apple/Application and File menus would have been disabled, `doEnableDisableMenus` is called to enable those menus.

doOpenFile

`doOpenFile` is called from `doOpenCommand` and from the Open Documents Apple event handler. The received file specification structure and file type are passed in a call to `doNewDocWindow` to open a window and create a `TXNObject`. Since the file system specification structure will be passed in the call to `TXNNewObject` in `doNewDocWindow`, `TXNNewObject` will read in the file and display its contents.

doCloseWindow

When `doCloseWindow` is closed to close a window, `TXNDeleteObject` is called to delete the `TXNObject` and all associated data structures. The window is then disposed of, and the global variable, which keeps track of the number of open windows, is decremented. If no windows remain open, `doEnableDisableMenus` is called to disable all of the applications windows less the Apple/Application and File menus.

doWriteFile

`doWriteFile` is called by `doSaveCommand` and `doSaveAsCommand`. As in the demonstration program Files, a "safe save" procedure is used to save files.

The three calls to `GetWindowProperty` retrieve the `TXNObject`, file system specification, and file type stored in the window object. At the next line, the encoding is set to Mac OS Encoding if the file type is '`TEXT`', otherwise encoding is set to Unicode.

After the temporary file is created and its data fork is opened, and if the file type is of type '`TEXT`', the resource fork is also created and opened.

The call to `TXNSave` then writes the contents of the document to the temporary file. Note that the file type is passed in the second parameter. If the file is of type '`TEXT`', `kTXNMultipleStylesPerTextDocumentResType` passed in the third parameter ensures that style information will be saved to the resource fork as a '`styl`' resource.

The call to `FSpExchangeFiles`, swaps the files' data by changing the information in the volume's catalog. The temporary file is then deleted, following which the file's data and resource forks are closed.

If this is a new file, `doCopyResources` is called to copy the missing application name string resource from the resource fork of the application file to the resource fork of the new file.